Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Curso de Engenharia de Software

Camada de Modelo do Sistema Educacenso para Software de Gestão Escolar

Autor: Henrique Pereira de Jesus Santos
Orientador: Prof. Dr. Maurício Serrano

Brasília, DF
2014

Henrique Pereira de Jesus Santos

Camada de Modelo do Sistema Educacenso para Software de Gestão Escolar

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB
Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Maurício Serrano

Co-Orientador: Prof. Dra. Milene Serrano

Brasília, DF
2014

Santos, Henrique.
Camada de Modelo do Sistema Educacenso para Software de Gestão Escolar / Henrique Pereira de Jesus Santos. Brasília: UnB, 2014.

Trabalho de Conclusão de Curso – Universidade de Brasília
Faculdade do Gama, Brasília, 2014. Orientação: Maurício Serrano

1. Educacenso. 2. Padrão MVC. 3. Censo Escolar I. Dr. Maurício Serrano. II. Camada de Modelo do Sistema Educacenso para Software de Gestão Escolar.

Camada de modelo do sistema Educacenso para software de gestão escolar

HENRIQUE SANTOS

Monografia submetida como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software da Faculdade UnB Gama - FGA, da Universidade de Brasília, em 25 de junho de 2014, apresentada e aprovada pela banca examinadora abaixo assinada:

Prof. Dr.: Maurício Serrano, UnB/ FGA
Orientador

Profa. Dra.: Milene Serrano, UnB/ FGA
Co-orientadora

Profa. Msc.: Fabiana Freitas Mendes, UnB/ FGA
Membro Convidado

Prof. Dr.: Paulo Roberto Miranda Meirelles, UnB/ FGA
Membro Convidado

Brasília, DF
2014

Aos meus pais, que me ensinaram a colocar Deus em todos os meus planos. Ao meu irmão Thiago por me incentivar a cursar Engenharia de Software e ao Dieisson por me mostrar que na vida, apesar das dificuldades, é possível ser feliz.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus, pois sei que Ele traçou em minha vida muitas oportunidades, e tem me ajudado a fazer as escolhas certas.

Ao meu orientador Dr. Maurício Serrano, por me indicar os passos que eu precisei seguir neste Trabalho de Conclusão de Curso.

A minha co-orientadora Dra. Milene Serrano, por me indicar como formalizar este trabalho, bem como incentivar a sua continuidade.

A professora Dra. Rejane Figueiredo, por me dar oportunidades desde o início da graduação e por me ajudar a escrever artigos melhores, respeitando padrões e regras.

A coordenadora-geral do Censo Escolar da Educação Básica, Célia Gedeon, por me acolher na equipe desde o primeiro dia que dela fiz parte.

Aos colegas Marcos Rogério, Ramon Borges, Gedalias Filho, Glauco Rocha e Jéferson Rosa, por estarem comigo na equipe que trabalha com o banco de dados do Censo Escolar.

Aos meus pais, por sempre me incentivarem a aproveitar as oportunidades e a colocar Deus em todas as minhas escolhas.

Ao meu irmão Thiago, por sempre me auxiliar em questões de programação e por sempre me incentivar nas escolhas.

Ao Dieisson, pela companhia, pelo apoio e pelos ensinamentos que me deram mais disciplina, calma e entendimento das boas coisas da vida.

Aos amigos da faculdade e colegas que estão comigo nesta jornada.

"Desistir... eu já pensei seriamente nisso, mas nunca me levei realmente a sério; é que tem mais chão nos meus olhos do que o cansaço nas minhas pernas, mais esperança nos meus passos, do que tristeza nos meus ombros, mais estrada no meu coração do que medo na minha cabeça."

Cora Coralina

Resumo

O Censo Escolar da Educação Básica é o maior levantamento de dados escolares do país e serve como base para diversos programas e formulação de políticas públicas. Atualmente, o processo percorrido para se informar dados ao censo é muito extenso e trabalhoso, devido ao fato de existirem muitas regras que devem ser respeitadas ao se responder o censo. Além disso, muitas escolas não possuem sistema próprio de gestão de dados escolares. Este Trabalho de Conclusão de Curso tem o objetivo de apresentar uma justificativa teórica, um processo para implementação de uma camada de modelo do sistema Educacenso para software de gestão de dados escolares, bem como os resultados obtidos com a implementação do projeto. Esta camada servirá de insumo para construção de sistemas que estejam compatíveis com as diversas regras instituídas pelo Censo Escolar, realizado pelo Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. Os sistemas implementados com o uso desta camada ajudarão escolas e secretarias de educação a gerenciar os seus dados e a informá-los ao censo escolar no período especificado pela lei. Além disso, espera-se contribuir com uma diminuição dos custos de contratação de serviços de desenvolvimento de software, possibilitar um aumento da fidedignidade dos dados informados ao censo, bem como facilitar o período de coleta devido à diminuição da recorrência de erros relacionados às regras de negócio do censo escolar.

Palavras-chave: Censo Escolar; Sistema Educacenso; Camada de modelo.

Abstract

The School Census in Brazil is the largest survey of school data in the country. It collects data from pre-school to secondary school and serves as the basis for various programs and public policy makers. The current process used to inform data to the Census is very complex and laborious due to the fact that many schools do not have a proper system of school data management and there are many rules that must be followed when responding to the Census. This work aims to provide a theoretical justification and process for implementing a model layer of the system Educacenso for software of school data management and the results achieved after the project's implementing. This layer will serve as input for building systems that are compatible with the various rules imposed by the School Census, which is conducted annually by the National Institute for Educational Studies Anisio Teixeira. The systems implemented using this layer will help schools and education departments to manage their data and report the information to the School Census within the period determined by law. In addition, we hope to contribute to diminishing the cost of hiring the services of software development, as well as enabling increased reliability of data reported to the Census, and facilitate the collection period due to a decrease in recurring errors related to business rules of the School Census.

Keywords: School Census, System Educacenso, Layer model

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

AEE	Atendimento Educacional Especializado
AUDIN	Auditoria Interna
CGACGIES	Coordenação-Geral de Avaliação dos Cursos de Graduação e Instituições de Ensino Superior
CGCEB	Coordenação-Geral do Censo da Educação Básica
CGCES	Coordenação-Geral do Censo da Educação Superior
CGCQES	Coordenação-Geral de Controle de Qualidade da Educação Superior
CGCQTI	Coordenação-Geral de Controle Qualidade e Tratamento de Informação
CGEC	Coordenação – Geral de Exames para Certificação
CGENADE	Coordenação-Geral do ENADE
CGIE	Coordenação-Geral de Informações e Indicadores Educacionais
CGIM	Coordenação-Geral de Instrumentos e Medidas
CGIME	Coordenação-Geral de Instrumentos e Medidas Educacionais
CGIS	Coordenação-Geral de Infraestrutura e Serviços
CGOFC	Coordenação-Geral de Orçamento, Finanças e Contabilidade
COGEP	Coordenação-Geral de Gestão de Pessoas
CGRL	Coordenação-Geral de Recursos Logísticos
CGSI	Coordenação-Geral de Sistemas de Informação
CGSNAEB	Coordenação-Geral do Sistema Nacional de Avaliação da Educação Básica
CIBEC	Centro de Informação e Biblioteca em Educação
DAEB	Diretoria de Avaliação da Educação Básica

DAES	Diretoria de Avaliação da Educação Superior
DEED	Diretoria de Estatísticas Educacionais
DGP	Diretoria de Gestão e Planejamento
FNDE	Fundo Nacional de Desenvolvimento da Educação
HQL	*Hibernate Query Language*
IDEB	Índice de Desenvolvimento da Educação Básica
IDE	*Integrated Development Environment*
INEP	Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira
JPA	*Java Persistence API*
LDB	Lei das Diretrizes Básicas da Educação
LIE	Layout de Importação/Exportação
MEC	Ministério da Educação
MOR	Modelagem Objeto Relacional
MVC	*Modeling-View-Controller*
OCDE	Organização para Cooperação e Desenvolvimento Econômico
PDE	Plano de Desenvolvimento da Educação
PNAE	Programa Nacional de Alimentação Escolar
PROJUR	Procuradoria Federal.
SAEB	Sistema de Avaliação da Educação Básica
SGBD	Sistema Gerenciador de Banco de Dados
TCC	Trabalho de Conclusão de Curso

TDD	Test Driven Development
XML	*Extensible Markup Language*

Sumário

1. Introdução

O Censo Escolar da Educação Básica é o maior levantamento de dados escolares do país. É utilizado como base para diversos estudos educacionais, bem como para distribuição de recursos feita pelo FNDE (Fundo Nacional de Desenvolvimento da Educação) e para programas governamentais na área da educação como Dinheiro Direto na Escola (PDDE), e o Programa Nacional de Alimentação Escolar (PNAE). Para a realização do censo, utiliza-se o sistema Educacenso, onde são cadastrados os dados escolares, que basicamente envolvem dados de escolas, docentes, alunos e turmas.

Uma das atribuições do Ministério da Educação (MEC) é a de realizar avaliações e pesquisas educacionais, conforme estabelecido pelo decreto nº 6.320/1930. O Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP), órgão da administração indireta do MEC, é responsável pela organização e manutenção das informações e estatísticas educacionais (BRASIL, 1997).

A Coordenação Geral do Censo da Educação Básica (CGCEB), uma das subdivisões da Diretoria de Estatísticas Educacionais (DEED) do INEP, é responsável pelo planejamento, coordenação e controle do processo que visa à realização do Censo Escolar (INEP, 2011).

Todos os anos, conforme o decreto nº 6425/2008, o INEP deve realizar o Censo Escolar, que é uma pesquisa declaratória que visa levantar dados escolares. Os estabelecimentos públicos e privados de educação básica são obrigados a fornecerem as informações solicitadas por ocasião do censo (BRASIL, 2008).

Para a realização do Censo, o INEP disponibiliza o Educacenso, que é um sistema informatizado de levantamento de dados, o qual utiliza ferramentas web para coleta, organização, transmissão e disseminação dos dados censitários, mediante o cruzamento de informações.

Nos últimos anos, secretarias estaduais e municipais de educação, bem como estabelecimentos públicos e privados, têm procurado informatizar seus dados escolares. Aqueles que já possuem um sistema buscam adaptá-lo ao Educacenso, de modo que o fornecimento de dados ao Censo Escolar seja mais facilitado e que a fidedignidade das informações seja maior.

Entidades que possuem sistema próprio buscam adaptar seus softwares ao Educacenso por meio da implementação das regras dispostas em um Layout de Importação/Exportação (LIE), disponibilizado pelo INEP. Porém, muitas vezes, no momento de se responder o Censo, estados, municípios e escolas se deparam com diversos erros ao se cadastrar dados, devido a falhas em seus sistemas ou a diferenças nas regras de cada sistema.

Este Trabalho de Conclusão de Curso visa aumentar a aderência dos sistemas escolares utilizados pelas diversas entidades de educação ao sistema Educacenso, bem como facilitar o fornecimento de dados ao Censo Escolar, aprimorar os testes realizados nos sistemas escolares em suas fases de transição e diminuir os custos relativos à contratação de empresas privadas para construção de novos sistemas.

Para que tais objetivos sejam alcançados, foi planejada e construída uma biblioteca de software que busca estar de acordo com as regras de negócio implementadas no sistema Educacenso. Esta biblioteca poderá ser distribuída pelo INEP aos estados e entidades de educação para que eles construam seus sistemas utilizando-a, aumentando assim a semelhança dos softwares escolares ao sistema Educacenso, e alcançando outros objetivos.

1.1. Motivação

O Educacenso é um sistema que não está integrado aos sistemas escolares existentes. Dessa forma, para se responder o Censo, a escola que não possui sistema deve reunir todas as informações necessárias e informa-las via sistema on-line, por meio do preenchimento de diversos formulários. A escola que possui sistema próprio pode gerar um arquivo em formato *.txt* no modelo estipulado pelo LIE, e submeter o arquivo no Educacenso, em um processo conhecido como migração.

Atualmente, muitos Estados e entidades de educação utilizam o sistema on-line para responder ao Censo, o que demanda muito tempo aos gestores escolares e aos responsáveis pelo Censo Escolar dos estados e municípios. Conforme consulta a base de dados do Censo Escolar 2013, apenas 29,09% das escolas foram informadas ao Censo por meio de migração (INEP, 2013).

A não aderência dos sistemas escolares ao sistema Educacenso gera erros recorrentes, que atrapalham no período de coleta. Com base nos dados do Censo Escolar 2012, todas as escolas que foram migradas tiveram algum erro antes de conseguir migrar a escola

corretamente, consequência de falhas em sistemas escolares na hora de se gerar o arquivo de migração e da não aderência desses sistemas às regras descritas no LIE.

A disponibilização de uma biblioteca em que as regras do Educacenso estão implementadas pode facilitar a construção de sistemas novos e a adaptação de sistemas já existentes ao Educacenso. Além disso, custos de construção de software de gestão escolar poderão ser diminuídos devido ao fato de a camada de modelo da aplicação já estar disponível.

1.2. Objetivos

Esta seção apresenta os objetivos gerais e específicos deste Trabalho de Conclusão de Curso.

1.2.1. Objetivo Geral

Desenvolver uma biblioteca em que as regras de negócio do sistema Educacenso apresentadas no Layout de Importação/Exportação estejam implementadas.

1.2.2. Objetivos Específicos

Para se alcançar o objetivo geral deste trabalho, os seguintes objetivos específicos foram definidos:

- Utilizar as tabelas do banco de dados do Educacenso para a construção de classes java, diminuindo custos de implementação;
- Utilização do Padrão MVC (*Model*, *View* e *Control*), para construção da biblioteca como camada de modelo;
- Utilização do TDD (Test Driven Development) para implementação dos métodos necessários para responder às regras de negócio do sistema, as quais estão apresentadas no Layout de Importação/Exportação; e
- Aplicar refatoração sobre a implementação da biblioteca com o intuito de garantir seus benefícios no código.

1.3. Metodologia

Dado o objetivo deste trabalho, desenvolver uma biblioteca em que as regras de negócio do sistema Educacenso, apresentadas no Layout de Importação/Exportação estejam implementadas, a metodologia foi, quanto aos fins, classificada como aplicada, justificando-se pela necessidade de se resolver os problemas descritos na Seção 1.1. Quanto aos meios: pesquisa bibliográfica, usada para explorar o assunto e delimitar o trabalho, e pesquisa documental, para a coleta de documentos que regulamentam o Censo Escolar e que descrevem seus processos e regras de negócio do sistema Educacenso.

Neste trabalho, foram levantadas as leis que regulamentam o Censo Escolar no Brasil, o LIE e padrões, metodologias e ferramentas de desenvolvimento de software a serem utilizados na construção da biblioteca. A partir da documentação do banco de dados do sistema Educacenso, foram geradas classes java para a biblioteca, onde posteriormente foram implementadas as regras de negócio do sistema Educacenso, expostas no LIE.

1.4. Organização do Trabalho

Este primeiro capítulo se destinou a apresentar o contexto do trabalho, bem como a motivação e objetivos que norteiam o desenvolvimento deste TCC. O texto seguinte está organizado segundo a estrutura abaixo:

Capítulo 2 – Apresenta um referencial teórico, tratando da origem e das atribuições do Ministério da Educação (MEC) e do Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP). Depois, apresentará a Diretoria de Estatísticas Educacionais (DEED), sua subdivisão, Coordenação Geral do Censo da Educação Básica (CGCEB) e o contexto do Censo. Ao final, será mostrado o processo básico do Censo, o funcionamento da ferramenta utilizada no levantamento e a estrutura do documento que descreve as regras de negócio do sistema.

Capítulo 3 – Apresenta o padrão MVC e a abordagem do TDD, os quais serão utilizados para o desenvolvimento do trabalho, trata sobre a modelagem objeto relacional, seu processo básico e suas atividades, os quais serão seguidos para a implementação da camada de modelo da aplicação proposta e apresenta as IDEs que permitem a realização das atividades indicadas pelo MVC, pelo TDD e pela modelagem objeto relacional.

Capítulo 4 – Apresenta a proposta do trabalho, bem como o estudo inicial realizado sobre o banco de dados do sistema Educacenso, as tarefas realizadas para geração de classes a partir do banco de dados, análise do código gerado e metodologia de desenvolvimento utilizada no trabalho.

Capítulo 5 – Apresenta os resultados da Fase 1 do projeto, destacando os resultados das engenharias reversas feitas sobre o sistema Educacenso e seu banco de dados.

Capítulo 6 – Apresenta os resultados da Fase 2 do projeto, destacando as *user stories* de cada *Sprint* do projeto, bem como os resultados de suas implementações, além dos testes finais feito sobre a camada de modelo e os aspectos de gerenciamento do projeto.

Capítulo 7 – Apresenta as considerações finais do trabalho.

2. Contexto do Censo Escolar da Educação Básica

Esta seção apresenta o contexto do Censo da Educação Básica no Brasil, expondo o histórico e atribuições dos principais órgãos envolvidos, a legislação que rege o levantamento, o sistema utilizado para a coleta de dados e a descrição do documento que traz as regras de negócio do sistema.

2.1. Ministério da Educação (MEC)

O Ministério da Educação foi criado em 1930 por meio do decreto n.º 19.402 logo que Getúlio Vargas assumiu o poder. O primeiro nome do órgão foi Ministério da Educação e Saúde Pública, e tinha como atribuição o desenvolvimento de atividades pertinentes a vários ministérios, tais como, saúde, esporte, educação e meio ambiente. Antes, os assuntos ligados à educação eram de competência do Ministério da Justiça.

Em 1953, o Ministério da Educação e Saúde Pública se tornou apenas Ministério da Educação e Cultura, com a sigla MEC. Já em 1961, com a aprovação da primeira Lei de Diretrizes e Bases da Educação (LDB), os órgãos estaduais e municipais ganharam mais autonomia, o que diminuiu a centralização do sistema educacional brasileiro no MEC. Em 1995, o órgão passou a ser responsável apenas pela área de educação.

Com base no decreto nº 6.320/2007, o Ministério da Educação tem como área de competência os seguintes assuntos:

i. Política nacional de educação;

ii. Educação infantil;

iii. Educação em geral, compreendendo ensino fundamental, ensino médio, ensino superior, ensino de jovens e adultos, educação profissional, educação especial e educação à distância, exceto militar;

iv. Avaliação, informação e pesquisa educacional;

v. Pesquisa e extensão universitária;

vi. Magistério; e

vii. Assistência financeira a famílias carentes para a escolarização de seus filhos ou dependentes.

Atualmente o MEC é composto por várias secretarias e por órgãos de Administração Indireta. Dentre eles, destaca-se o Instituto Nacional de Estudos e Pesquisas Educacionais, presente na estrutura organizacional do MEC, apresentada pela Figura 1.

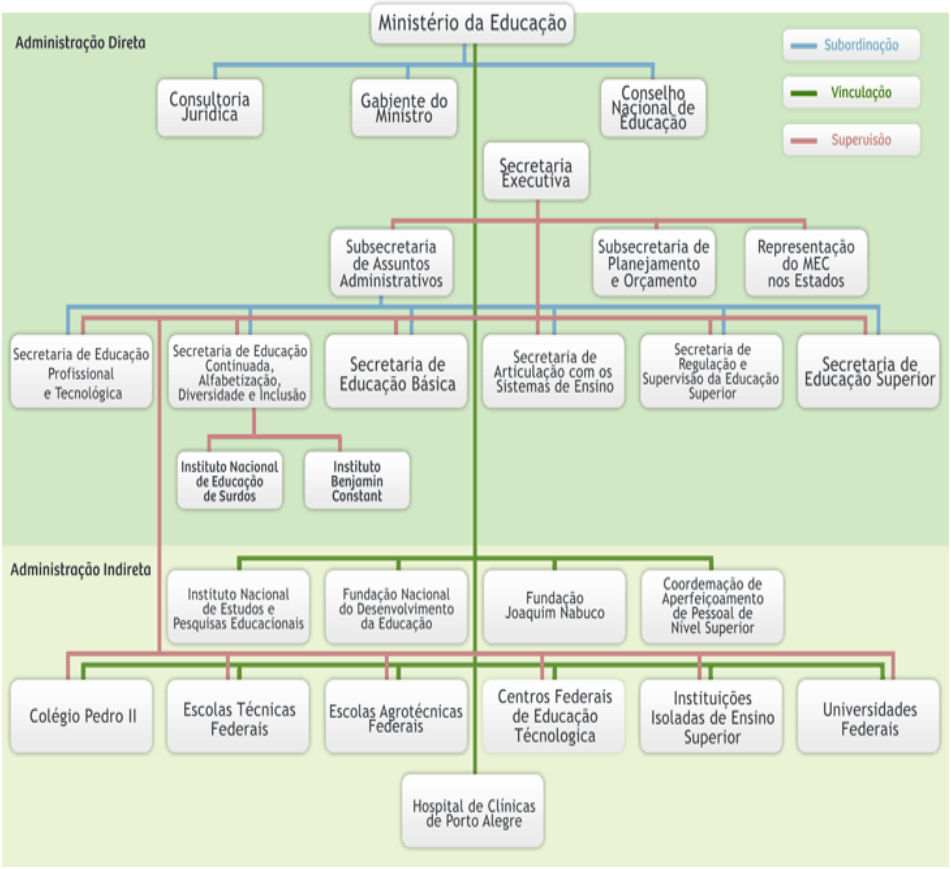

Figura 1 – Estrutura organizacional do MEC

2.2. Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP)

Um dos órgãos da Administração Indireta do MEC é o Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP). O INEP foi criado em 1937 e no ano seguinte o órgão iniciou seus trabalhos após a promulgação do Decreto-Lei nº 580, que regulamenta a sua organização e estrutura. Na época suas atribuições se resumiam a organizar a documentação relativa à história e ao estado atual das doutrinas e técnicas pedagógicas,

promover inquéritos e pesquisas e prestar assistência técnica, esclarecimentos e soluções aos problemas dos serviços estaduais, municipais e particulares de educação.

Em 1972, o INEP foi transformado em órgão autônomo, e passou a ser denominado como Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. Neste momento, seu objetivo passava a ser o de realizar levantamentos da situação educacional do País.

Atualmente, com base na Lei 9.448/1997, o INEP tem as seguintes atribuições:

i. Organizar e manter o sistema de informações e estatísticas educacionais;
ii. Planejar, orientar e coordenar o desenvolvimento de sistemas e projetos de avaliação educacional, visando o estabelecimento de indicadores de desempenho das atividades de ensino no País;
iii. Apoiar os Estados, o Distrito Federal e os Municípios no desenvolvimento de sistemas e projetos de avaliação educacional;
iv. Desenvolver e implementar, na área educacional, sistemas de informação e documentação que abranjam estatísticas, avaliações educacionais, práticas pedagógicas e de gestão das políticas educacionais;
v. Subsidiar a formulação de políticas na área de educação, mediante a elaboração de diagnósticos e recomendações decorrentes da avaliação da educação básica e superior;
vi. Coordenar o processo de avaliação dos cursos de graduação, em conformidade com a legislação vigente;
vii. Definir e propor parâmetros, critérios e mecanismos para a realização de exames de acesso ao ensino superior;
viii. Promover a disseminação de informações sobre avaliação da educação básica e superior;
ix. Articular-se, em sua área de atuação, com instituições nacionais, estrangeiras e internacionais, mediante ações de cooperação institucional, técnica e financeira bilateral e multilateral.

Nos últimos anos o INEP reorganizou o sistema de levantamentos estatísticos e suas atividades foram focadas em avaliações de todos os níveis educacionais. Com base no Decreto nª 6.317 de 20 de dezembro de 2007, o INEP possui seis diretorias. Entre elas, destaca-se a Diretoria de Estatísticas Educacionais (DEED), responsável pela realização do Censo da Educação Básica e da Educação Superior. A DEED é apresentada na estrutura organizacional do INEP, mostrada na Figura 2.

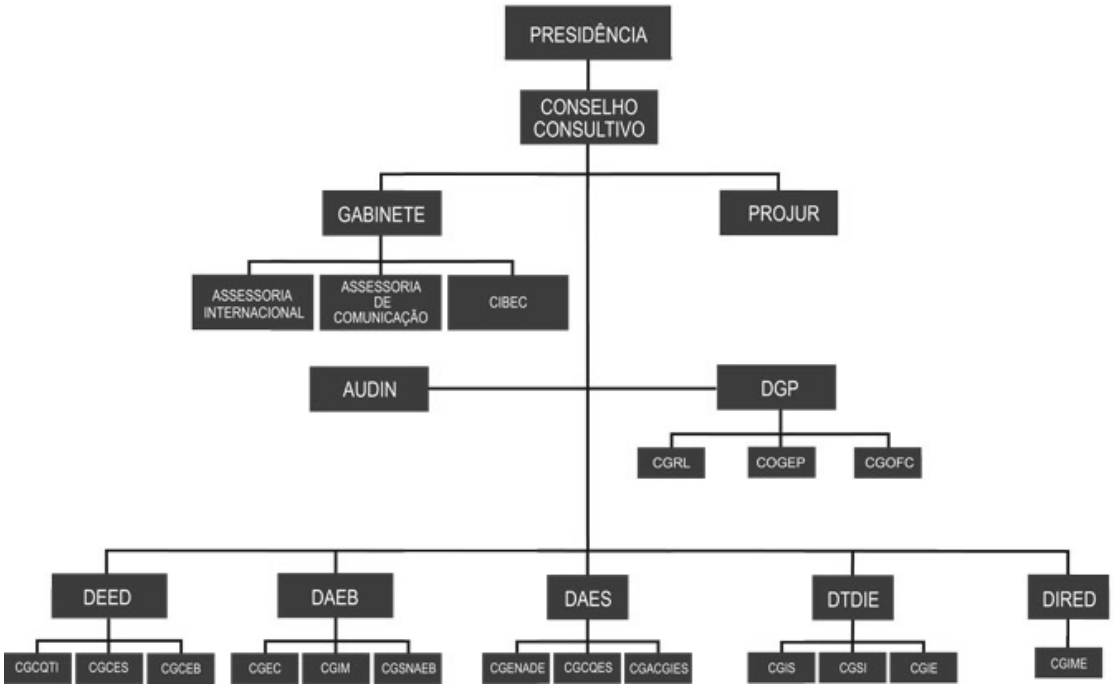

Figura 2 – Estrutura Organizacional do INEP

2.2.1. Diretoria de Estatísticas Educacionais - DEED

Conforme informado na Seção 2.2, a DEED é responsável pela realização de dois censos: da educação básica e da educação superior. A DEED é composta por três coordenações, duas delas responsáveis pelos censos e outra responsável pelo tratamento da informação. Essas coordenações possuem mais subdivisões, conforme a Figura 3.

Dentre as três coordenações da DEED, destaca-se a Coordenação-Geral do Censo Escolar da Educação Básica (CGCEB), por ser responsável pela execução do processo necessário para a realização do Censo Escolar da Educação Básica, cujo sistema de coleta é alvo de estudo deste trabalho. Com base na proposta de restruturação da DEED (INEP, 2011), a CGCEB possui as seguintes atribuições:

i. Planejar, coordenar e controlar ações voltadas à produção de dados estatísticos da educação básica, no âmbito de suas coordenações;

ii. Definir e propor parâmetros, critérios, estratégias e mecanismos para a coleta de dados da educação básica, no âmbito de suas coordenações;

iii. Promover a articulação institucional com os órgãos do MEC, para identificar as demandas específicas de cada área, no que se refere às necessidades de informações estatísticas da educação básica;

iv. Coordenar as ações relativas ao levantamento de dados, adotando mecanismos de acompanhamento e avaliação junto às secretarias estaduais de educação e/ou outros órgãos envolvidos; e

v. Definir os mecanismos de coleta de dados necessários à produção de informações estatísticas do Censo Escolar.

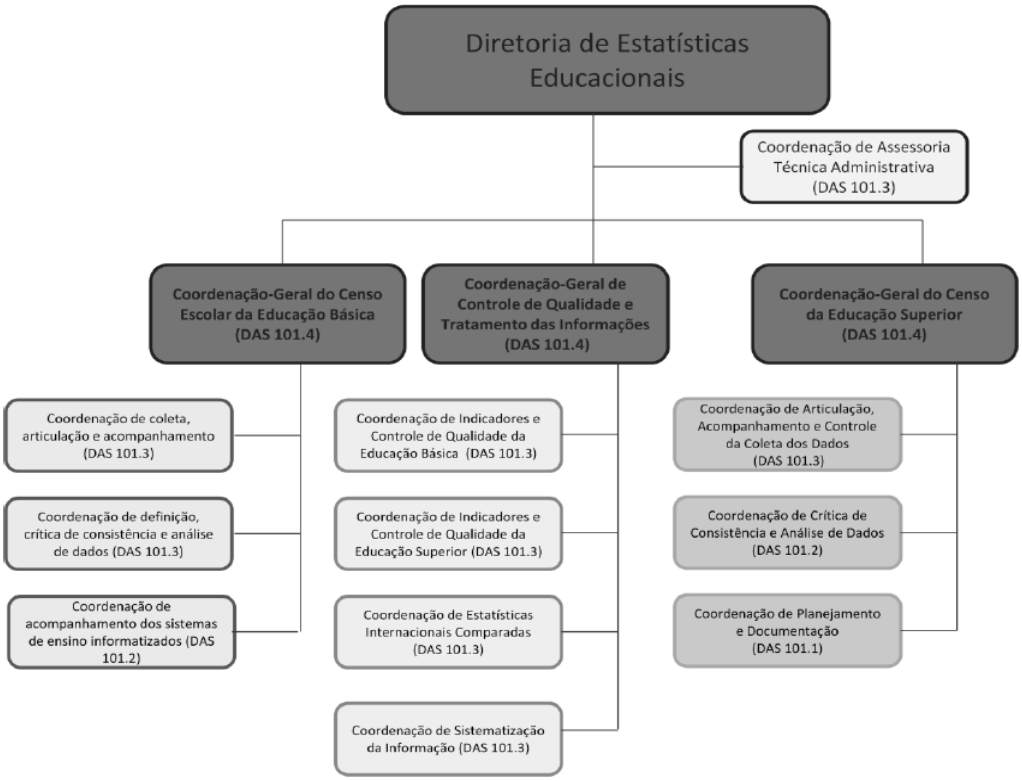

Figura 3 – Estrutura Organizacional da Diretoria de Estatísticas Educacionais

2.3. Censo da Educação Básica

O Censo Escolar da Educação Básica é uma pesquisa declaratória realizada todos os anos pelo INEP (INEP, 2013). Os estabelecimentos públicos e privados de educação básica são obrigados a fornecer as informações solicitadas por ocasião do Censo (BRASIL, 2008).

Conforme o Resumo Técnico do Censo da Educação Básica (INEP, 2013)

> O Censo é o mais relevante e abrangente levantamento estatístico sobre a educação básica do Brasil. Os dados coletados são fonte completa de informações que o MEC utiliza para formular políticas públicas, desenhar programas educacionais e definir critérios para atuação supletiva do MEC às escolas, aos Estados e aos municípios.

Além disso, o Censo subsidia o cálculo do Índice de Desenvolvimento da Educação Básica (Ideb), que é referência para que sejam traçadas as metas do Plano de Desenvolvimento da Educação (PDE). A data de referência para as escolas informarem seus dados educacionais ao Censo Escolar é a última quarta-feira do mês de maio.

2.4. Sistema Educacenso

"O Educacenso é um sistema informatizado de levantamento de dados do Censo Escolar (INEP, 2013). É implementado nas linguagens de programação PHP 5 (módulo on-line) e Java (módulo de migração) com servidor de aplicação Apache e JBoss, respectivamente" (PINHEIRO at al. 2006). Utiliza ferramentas web para a coleta, organização, transmissão e disseminação dos dados censitários, mediante o cruzamento de informações de quatro cadastros de dados:

- Cadastro de escola: abrange informações que dizem respeito à infraestrutura da escola (local de funcionamento, salas, tipo de abastecimento de água e de energia elétrica), suas dependências (diretoria, secretaria, cozinha), equipamentos de multimídia, etapas e modalidades de escolarização oferecidas, dependência administrativa e outras.

- Cadastro de aluno: abrange informações como sexo, cor/raça, idade, etapa e modalidade de ensino frequentada, nacionalidade, endereço, entre outras, de cada aluno.

- Cadastro de profissional escolar em sala de aula: abrange informações como sexo, cor/raça, idade, escolaridade, etapa e modalidade de ensino de exercício, turmas de exercício, disciplinas que ministra, e outras, de cada profissional escolar.

- Cadastro de turma: abrange informações, tais como, o nome, o tipo da turma (escolarização, atividade complementar, classe hospitalar, e outras), horários de início e de término das aulas, modalidade e etapa de ensino, disciplinas e dias da semana nos quais a turma funciona.

"O preenchimento dos dados é feito diretamente na Internet, por meio do sistema Educacenso. Com o cpf e senha cadastrada, o usuário de cada escola deve acessar o sistema para informar os dados ao Censo" (INEP, 2013). Existem duas formas de se responder o Censo Escolar por meio do Educacenso: por sistema on-line ou por migração. Quando se

utiliza o sistema on-line para responder o Censo, o usuário irá preencher diversos formulários com os dados de escola, profissional escolar, turma e aluno. A Figura 4 demonstra parte de uns dos formulários que deve ser preenchido para cadastro de aluno:

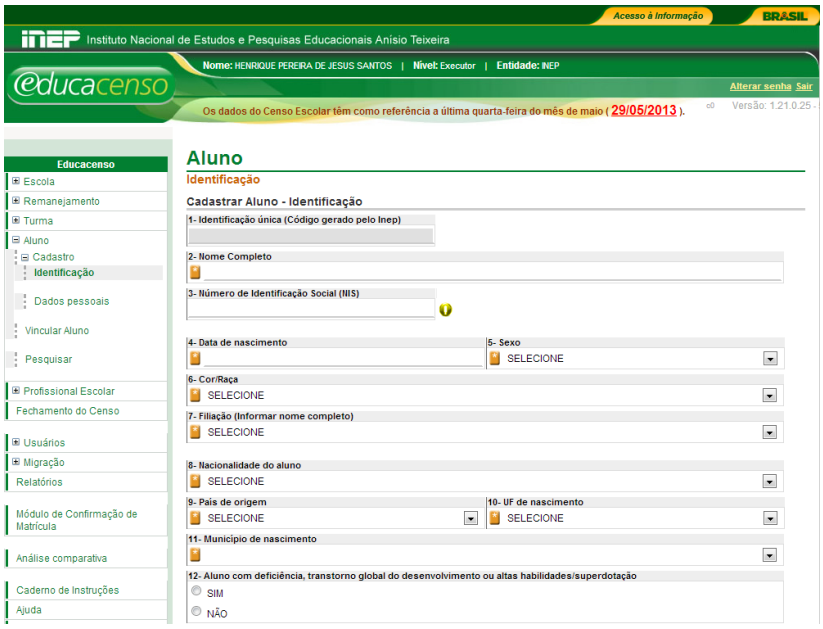

Figura 4 – Cadastro de Aluno no Educacenso

Caso o usuário opte por informar o Censo via migração, ele terá de gerar um arquivo em formato especificado pelo INEP, o qual será detalhado na Seção 2.5. Geralmente estes arquivos de migração são gerados pelo sistema próprio da escola, do município ou do Estado. Este arquivo será submetido a algumas verificações, feita em etapas. A primeira etapa de verificação é feita por um sistema conhecido como Migradados, que também é responsável por enviar o arquivo para o sistema. O Migradados é disponibilizado ao usuário no próprio sistema Educacenso, na opção do menu "Migração" conforme mostra a Figura 5.

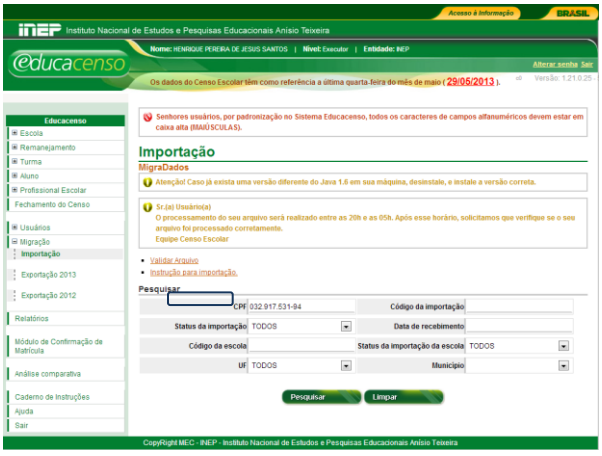

Figura 5 - Migração

Ao clicar no link “Validar Arquivo”, é iniciado o download do Migradados, o qual necessita do Java 6 ou superior para ser executado. A Figura 6 mostra o Migradados, onde o arquivo de migração é validado.

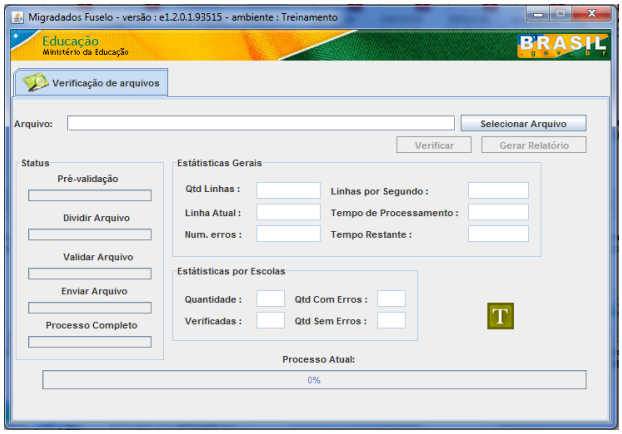

Figura 6 – Migradados

Quando um arquivo é validado com sucesso, o Migradados mostra a mensagem apresentada pela Figura 7 à esquerda. Quando o arquivo possui erros, o Migradados mostra a mensagem apresentada pela Figura 7 à direita e gera um relatório .pdf que indica os erros.

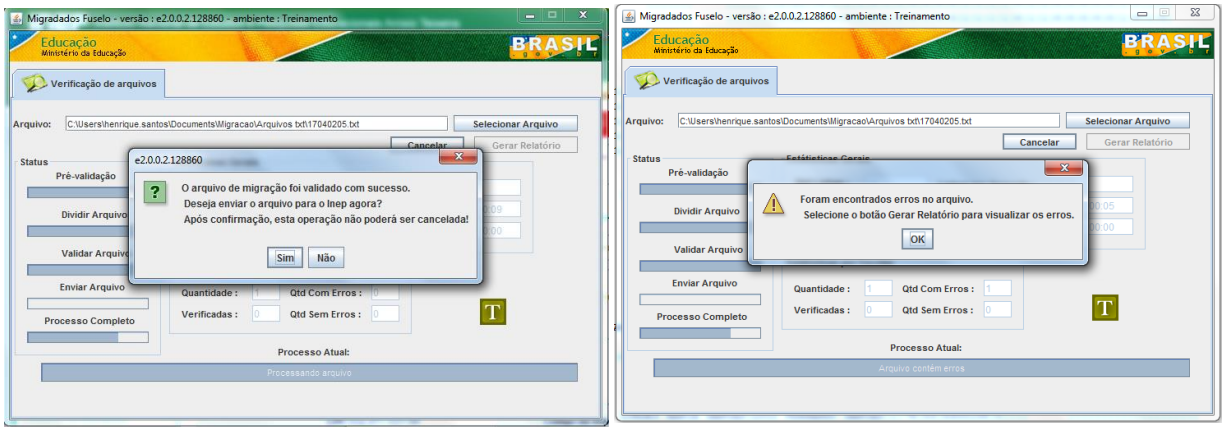

Figura 7 – Arquivo validado com sucesso à esquerda e arquivo com erros à direita

Após o arquivo ser validado pelo Migradados e enviado ao Educacenso, o usuário terá de acompanhar o status do arquivo e depois proceder com o fechamento do Censo da sua escola.

2.5. Layout do Arquivo de Importação/Exportação

O Layout do Arquivo de Importação/Exportação (LIE) é um documento elaborado pela CGCEB que apresenta a estrutura do arquivo texto com os dados que serão migrados

para o sistema Educacenso. Além de apresentar esta estrutura, este layout tem a função de apresentar as regras de cada campo de dado, englobando o tamanho, a obrigatoriedade, o formato, as regras, as condicionalidades e as validações.

A estrutura básica de um arquivo de migração é composta por dez registros. Cada registro é responsável por agrupar um conjunto de dados específico. Com base no Layout de Importação/Exportação (2013), os registros são:

- Registro 00 (Cadastro de Escola – Identificação) – Este registro é responsável por agrupar os dados referentes ao nome da escola, situação de funcionamento, início e término do ano letivo, endereço, telefone, dependência administrativa, entre outros;

- Registro 10 (Cadastro de Escola – Caracterização e Infraestrutura) – Este registro agrupa dados referentes ao gestor escolar, local de funcionamento da escola, abastecimento de água, energia elétrica e esgoto, destinação do lixo, dependências e equipamentos existentes na escola, modalidades e etapas de ensino oferecidas, entre outros;

- Registro 20 (Cadastro de Turma) – Agrupa dados referentes ao código e nome da turma, horário de início e de término, dias da semana, modalidade e etapa de ensino e disciplinas ministradas;

- Registro 30 (Cadastro de Profissional Escolar em Sala de Aula – Identificação) – Agrupa dados referentes aos docentes, tais como código, nome, sexo, local de nascimento, possíveis deficiências, entre outros;

- Registro 40 (Cadastro de Profissional Escolar em Sala de Aula – Documentos e Endereço) – Agrupa dados referentes aos docentes, tais como, cpf, endereço, entre outros;

- Registro 50 (Cadastro de Profissional Escolar em Sala de Aula – Dados Variáveis) – Agrupa dados ainda referente aos docentes, que dizem respeito à sua escolaridade e formação superior;

- Registro 51 (Cadastro de Profissional Escolar em Sala de Aula – Dados de Docência) – Agrupa dados de docência, que informam a turma em que o docente é vinculado e as disciplinas as quais ele leciona;

- Registro 60 (Cadastro de Aluno – Identificação) – Agrupa dados referentes aos alunos, tais como código, nome, sexo, local de nascimento, possíveis deficiências, recursos necessários para a participação do aluno em avaliações do INEP (Prova Brasil, Saeb, etc.) entre outros;

- Registro 70 (Cadastro de Aluno – Documentos e Endereço) – Agrupa dados de documentação do aluno, tais como, RG, NIS, CPF, certidão de nascimento, entre outros e seus dados de endereço; e

- Registro 80 (Cadastro de Aluno – Vínculo (Matrícula)) – Agrupa dados referentes ao vínculo (matrícula) dos alunos em turmas e dados de utilização de transporte escolar público.

O LIE define a estrutura de cada registro, conforme o modelo mostrado na Tabela 1.

Tabela 1 – Estrutura dos Registros

Sequencia	**Campo**	**Fixo**	**Tamanho**	**Formato**	**Obrigatório**	**Regras**	**Condicionalidades e Validações**
Número do campo no registro.	Nome do campo.	Tamanho fixo ou não.	Número de caracteres que o campo pode ter.	Formato: Numérico ou Alfanumérico.	Indica se o campo é obrigatório ou não.	Regras gerais do campo.	Detalhamento das regras e outras condições e validações.

O LIE traz também diversas regras para os campos apresentados, e que devem ser respeitadas ao se responder o Censo. Elas estão descritas na coluna “Regras” de cada registro do LIE e são detalhadas na coluna “Condicionalidades e Validações”. A Figura 8 apresenta alguns exemplos de campos e regras do LIE.

Seq.	Campo	Fixo	Tamanho	Form	Obrig	Regras	Condicionalidades e Validações
8	Data de Nascimento	Sim	10	D	Sim	Deverá ser obrigatoriamente preenchido no seguinte formato: dd/mm/aaaa. A data de nascimento deverá ser uma data válida. O ano deverá ser anterior ou igual 2000 e posterior ou igual a 1919.	Deverá ser anterior ou igual à 2000. O ano informado deverá ser posterior ou igual a 1919 A data será validada apenas pelo ano.
9	Sexo	Sim	1	N	Sim	Valores permitidos: 1 - Masculino 2 - Feminino	
10	Cor/Raça	Sim	1	N	Sim	Valores permitidos: 0 - Não Declarada 1 - Branca 2 - Preta 3 - Parda 4 - Amarela 5 – Indígena	

Figura 8 – Exemplos de campos e regras do LIE

3. Padrões, Metodologias e Ferramentas

Esta seção apresenta o referencial teórico da engenharia reversa, da modelagem objeto relacional, do padrão MVC, da metodologia de desenvolvimento TDD e das ferramentas que serão utilizadas para o desenvolvimento deste TCC.

3.1. Engenharia Reversa

A engenharia reversa compreende um trabalho com um produto já existente, com a finalidade de entender seu funcionamento, seus objetivos e comportamento em determinadas circunstâncias (JUNIOR. SOUZA, et al. 2005). “É a avaliação sistemática de um produto com o propósito de fazer uma réplica. Este processo pode ter o objetivo de se produzir uma cópia ou de se incorporar melhoramentos em um projeto”. (ARONSON, 1996 apud FERNEDA 1999).

Sua utilização em sistemas de software ajuda a entender o funcionamento de um programa, bem como o seu comportamento, suas regras de negócios, arquitetura e outros. O entendimento propiciado através da engenharia reversa permite novas implementações de software que possua comportamento e regras de negócios semelhantes, para substituição do existente ou utilização do novo sistema em outros contextos.

Existem algumas dificuldades quando se realiza a engenharia reversa. Em termos de software, muitas vezes o sistema que se quer analisar possui uma vasta documentação e código-fonte disponível. Já em outros casos, não existe documentação do software ou o código-fonte não é aberto. Outros contextos e combinações podem ocorrer. Dessa forma, existem técnicas que podem ser aplicadas em alguns casos comuns.

Segundo Junior e Souza (2005), para se realizar a engenharia reversa quando o código-fonte não está disponível, podem-se usar os métodos de análise de fluxo de dados, que é uma análise da troca de informações do sistema, desassemblar, onde, com um desassembler consegue-se obter a linguagem de máquina diretamente do programa, e descompilação, onde, com o uso de um descompilador, tenta-se recriar o código-fonte em uma linguagem de alto nível, tendo disponível apenas o código de máquina. Já quando o código-fonte está disponível, os autores recomendam a análise estática e dinâmica do código, análise dos dados feita por meio de um estudo do banco de dados e estudo da documentação existente.

Considerando que este trabalho visa à construção de uma biblioteca em que as regras de negócio do sistema Educacenso expostas no LIE estejam implementadas, a engenharia reversa foi utilizada no estudo do sistema, dando foco ao seu funcionamento, aos seus fluxos principais e ao seu banco de dados. Ao se realizar a engenharia reversa do sistema Educacenso e do seu banco de dados, foi possível extrair informações sobre o processo de resposta ao censo, bem como sobre a estrutura das tabelas onde os dados são armazenados, tornando possível um melhor entendimento do sistema e um melhor projeto de implementação da biblioteca.

3.2. Modelagem Objeto Relacional

Sistemas de software devem, de alguma maneira, gerenciar a persistência de seus dados. Antes do surgimento dos Sistemas Gerenciadores de Banco de Dados (SGBDs), existia nos sistema um forte acoplamento entre a aplicação e o gerenciamento de persistência de dados, já que os programas criados tinham que, além de executar suas funções, controlar a armazenagem e extração de dados (SILBERSCHATZ, 2006 apud BENEGA, 2010).

Segundo Benega (2010), os primeiros SGBDs, bem como grande parte dos atuais utilizam o modelo relacional, armazenando os dados através de uma coleção de tabelas que se relacionam entre si. Cada tabela possui linhas não ordenadas, e que por sua vez possuem uma série de colunas. Cada coluna possui um nome, tipo e um formato para seus valores.

“O Modelo Orientado a Objetos aproxima o programador do mundo real, no qual tudo pode ser visto como objetos, como por exemplo, livro, aluno, faculdade” (DEITEL, 2006). Para Benega (2010), os objetos dentro da computação são unidades que encapsulam seu significado próprio e que possuem estado e comportamento.

Segundo Deitel (2006), para a implementação de um código mais legível e reutilizável, a programação orientada por objetos utiliza alguns conceitos em seu paradigma:

- Agregação – Um objeto contém outros objetos dentro de si. Existem dois tipos de agregação. O primeiro tipo, também nomeado agregação, consiste em um relacionamento entre dois objetos, onde um pode viver sem a existência de outro. O outro tipo, nomeado composição, consiste em um relacionamento onde um objeto necessita da existência do outro.

- Herança – Forma de reutilização de software onde se cria uma classe que absorve os dados e comportamentos de uma classe existente e os aprimora com novas capacidades. Com este recurso, pode-se modelar uma hierarquia de classes.

- Polimorfismo – Permite processar objetos de classes que fazem parte da mesma hierarquia de classes, como se todos fossem objetos da classe básica da hierarquia. Por exemplo, aluno e professor são objetos diferentes, porém ambos são pessoas. Dessa forma, é possível trabalhar com os dois tipos diferentes utilizando o tipo pessoa.

Para Bauer (2005), citado por Benega (2010)

> O modelo relacional não atende muito bem ao paradigma orientado por objetos, por não implementarem nativamente o conceito de herança utilizado pelos objetos. Muitas hierarquias, regras e relacionamentos tornam difícil a representação do modelo OO em um modelo simples de tabelas e relacionamentos.

Para não se ter a necessidade de alterar um banco de dados ou modelá-lo de forma mais adequada ao paradigma orientado por objetos, pode-se criar uma camada dentro da aplicação que mapeie os objetos com as tabelas (BENEGA, 2010).

Segundo Ambler (2003) citado por Torres (2009)

> A modelagem objeto relacional (MOR) busca a solução do problema de incompatibilidade da impedância no nível da implementação, permitindo ao desenvolvedor de sistemas de utilizar a tecnologia relacional e a programação orientada a objetos em conjunto, abstraindo a estrutura física do banco de dados.

Dessa forma, a MOR se trata de "ferramentas ou *frameworks* complexos que realizam o mapeamento do objeto no modelo relacional de forma automatizada e transparente" (BAUER, 2005 apud BENEGA, 2010).

Existem algumas especificações para que a MOR seja feita. Uma delas é o JPA (Java Persistence API), uma especificação da linguagem Java que define padrões para o comportamento e desenvolvimento de *frameworks* de MOR e de sistemas que utilizam seus serviços. Esses padrões são para anotações, arquivos de configuração e regras para escrita de código (TORRES, 2009).

Uma ferramenta de grande aceitação entre os desenvolvedores de sistemas orientados a objetos para a MOR e que implementa as especificações estabelecida pelo JPA é o

Hibernate. A configuração do *Hibernate* é feita por meio de arquivos XML ou por meio de *Annotations* (nova versão do *Hibernate),* que permite fazer anotações sobre o mapeamento em cada classe que se quer mapear no sistema. (FERNANDES, 2005 apud GUERRA, 2008).

O diagrama apresentado pela Figura 9 mostra as atividades que são percorridas pelo sistema para realizar o mapeamento objeto-relacional de uma classe para o banco de dados, por meio do *Hibernate.*

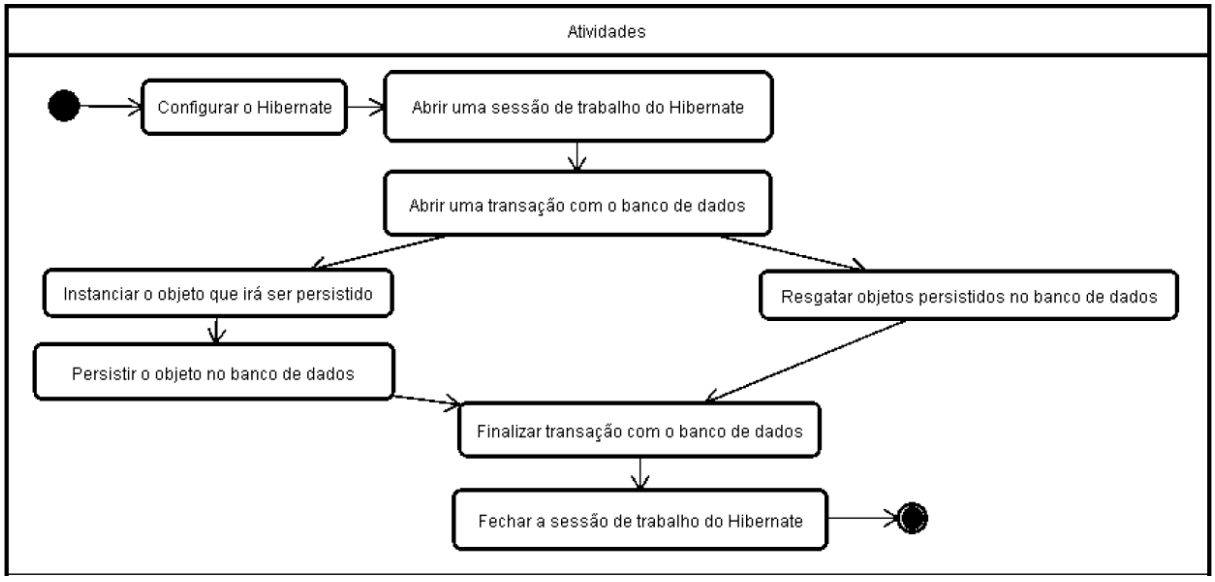

Figura 9 - Diagrama de Atividades de um mapeamento objeto-relacional.

Fonte: BAUER, 2007 apud GUERRA, 2008

A descrição das atividades expostas no diagrama apresentado pela Figura 7 se encontra na Tabela 2.

Tabela 2 – Atividades de um mapeamento objeto relacional.

Atividade	**Descrição**
Configurar o *Hibernate*	Ao iniciar uma classe que realiza um mapeamento objeto relacional, o sistema procura pelo arquivo XML que possui as informações necessárias para se configurar o *Hibernate.*
Abrir uma sessão de trabalho do *Hibernate*	Após ser configurado, o *Hibernate* abrirá uma sessão de trabalho, que corresponde a um classe que contém os métodos que realizam o mapeamento objeto relacional.

Abrir uma transação com o Banco de Dados	Uma transação com o banco de dados é aberta para que ocorram as operações no momento em que os métodos responsáveis por realizarem o mapeamento objeto relacional forem utilizados.
Instanciar classes	Deverá ser criada uma instância de uma classe quando o objetivo for de persistir um objeto.
Persistir o objeto	O objeto que foi instanciado na atividade descrita anteriormente é passado como parâmetro para o método que é responsável pela persistência de objetos. Neste momento o *Hibernate* cria um comando de inserção de dados e o executa no banco de dados.
Resgatar um objeto	O *Hibernate* utiliza HQL (*Hibernate Query Language),* uma linguagem própria para se resgatar objetos do banco de dados.
Finalizar transação com o banco de dados	As transações com o banco de dados são encerradas.
Fechar sessão de trabalho do *Hibernate*	Fechamento da sessão de trabalho do *Hibernate*, para que não haja prejuízo no desempenho do sistema.

Fonte: BAUER, 2007 apud GUERRA, 2008

Existem várias vantagens em se implementar o mapeamento objeto relacional. Para Esjug, (2007 apud GUERRA, 2008), algumas delas são:

- Manutenibilidade – Segundo Bauer (2005), o mapeamento objeto relacional é uma maneira de transformar dados da representação orientada por objetos para a representação de banco de dados relacional. Como é feito de maneira automática, pode-se reduzir o número de linhas de código fonte do sistema, diminuindo a complexidade de manutenção.

- Desempenho e produtividade na implementação - O tempo que se economiza no desenvolvimento devido à redução do número de linhas de código pode favorecer o desenvolvedor a otimizar e refatorar o código.

3.3. Padrão MVC

De acordo com Sommerville (2003) o padrão MVC aceita a apresentação dos dados de diferentes maneiras, permitindo assim múltiplas apresentações de um objeto e estilos separados de interação com cada uma delas. Dessa forma, quando um dado é modificado por meio de uma apresentação, todas as outras são atualizadas.

O MVC (*Modeling-View-Controller*) foi proposto em 1979 por Trygve Reenskaug, e é um padrão de arquitetura de projeto de software que divide o sistema em três tipos de camadas. Também define a forma de se integrar os componentes de cada camada com o objetivo de isolar a lógica de negócio da interface do usuário, aumentando a flexibilidade do sistema, a reutilização do código e a manutenção (SIMÕES; SANTOS, 2009).

O MVC é composto por três tipos de objetos: o modelo é o objeto de aplicação, ou seja, o domínio da informação; a visão, que é a apresentação do sistema (interface com o usuário); e controlador, que é a maneira como a interface do usuário reage às suas entradas (GAMMA et al, 2000 apud FONSECA, et al. s/d).

Para Reenskaug (1979), o modelo é uma representação ativa de uma abstração em forma de dados em um sistema de computação. Ela deve representar qualquer coisa de forma abstrata como uma representação do objeto real. Como exemplo de domínio, tem-se aluno, professor e turma como domínios de um sistema acadêmico. Além de representação, a camada de modelo é responsável por encapsular objetos, responder a consultas do banco de dados e notificar *views* de mudança.

A visão é a apresentação do modelo para o usuário. Ou seja, é a interface com o mundo exterior à aplicação. Já o controle é a camada que gerencia a comunicação entre a camada de modelo e de visão. Além disso, ela é responsável pelo controle do fluxo de dados, regras de negócios e ações dos usuários (FONSECA, et al. s/d). Dessa forma, a camada de controle, com base nas ações do usuário, comunica-se com a camada de modelo, que irá executar ações e atualizar a camada de visão com o resultado da solicitação.

A Figura 10 retrata o funcionamento do padrão MVC:

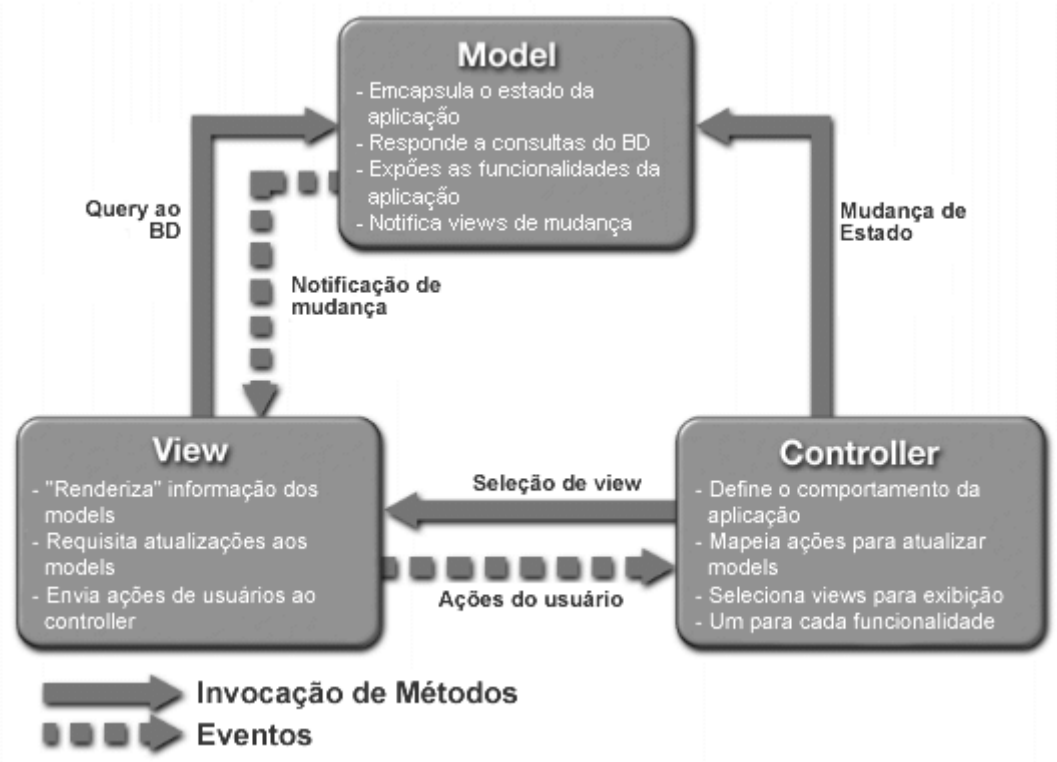

Figura 10 – Padrão MVC

Fonte: FONSECA, et al. s/d

Para Caetano (2013), as razões para se separar um sistema em camadas se deve ao fato de as mudanças mais frequentes acontecerem na interface com o usuário (visão) e, normalmente não exigem mudanças no processo de negócio (controle) e na modelagem de dados (modelo). Além disso, mudanças menores na modelagem de dados não exigem modificações no restante do sistema, assim como possíveis modificações no processo de negócio.

Dessa forma, segundo Caetano (2013), a separação facilita a manutenção, por centralizar as mudanças em partes específicas do sistema. Além disso, o MVC irá aumentar a possibilidade de reuso dos componentes desenvolvidos. Dessa forma, preserva-se o investimento de desenvolvimento feito no sistema.

3.4. SCRUM

O Scrum é uma metodologia de desenvolvimento de produtos complexos que vem sendo usada desde o início dos anos 90, e é fundamentado na teoria de controle de processos empíricos, empregando uma abordagem iterativa e incremental a fim de otimizar o controle de riscos e a previsibilidade em um projeto (SCHWABER; SUTHERLAND, 2011).

Para Schwaber e Sutherland (2011), o Scrum é um *framework* estrutural, não sendo um processo ou uma técnica para construir produtos e sim um *framework* dentro do qual se pode empregar vários processos ou técnicas. Consiste em equipes do Scrum associadas a papéis, eventos, artefatos e regras. Cada componente estabelecido pelo Scrum serve para um propósito específico.

Os times Scrum são auto organizáveis e multifuncionais e escolhem a melhor forma para executarem seus trabalhos. É composto pelos seguintes membros:

- *Product Owner* – Dono do projeto o qual é responsável por gerenciar o *Backlog* do Produto, que contém os itens ou funcionalidades do produto requerido;
- Equipe de Desenvolvimento – Composta pelos profissionais que realizam o trabalho a fim de entregar uma versão usável que potencialmente incrementa o produto ao final de cada *Sprint* do projeto; e
- *Scrum Master* – Pessoa responsável pela garantia de entendimento e aplicação do Scrum no projeto.

Como se pode notar na descrição dos membros dos times, o Scrum prevê alguns eventos que criam rotina e minimizam a necessidade de reuniões não definidas pelo *framework*. Esses eventos são descritos na Tabela 3.

Tabela 3 – Eventos do Scrum

Evento	Descrição
Sprint	Um *time-box* de um mês ou menos durante o qual uma versão incremental potencialmente utilizável do produto é criada. As versões criadas incrementam versões anteriores do produto, e este fato dá ao *framework* a característica de iterativo (vários *Sprints* para incrementar o produto*)* e incremental (uma versão complementa ou adiciona às funcionalidades de outra anterior).
Reunião de Planejamento da	Visa o planejamento do que será produzido em uma Sprint.

Sprint	
Reunião Diária	Evento de 15 minutos que têm o objetivo fazer com que a Equipe de Desenvolvimento sincronize as atividades e crie um plano para as próximas 24 horas.
Revisão da *Sprint*	Executada ao final da Sprint, tem o objetivo de inspecionar o incremento e adaptar o *Backlog* do Produto, se necessário.
Retrospectiva da *Sprint*	Visa permitir que o Time Scrum inspecione a si próprio e crie um plano para melhorias a serem aplicadas no *Sprint* seguinte.

Fonte: Schwaber e Sutherland, 2011

Além dos eventos, o Scrum prevê alguns artefatos:

- *Backlog* do Produto – Lista ordenada de tudo o que é necessário ao produto. Caracteriza um conjunto de requisitos do sistema, não necessariamente estável.

- *Backlog* do *Sprint* – Lista de itens do *Backlog* do Produto que serão desenvolvidas em uma *Sprint*.

Segundo Schwaber e Sutherland (2011), o Scrum somente existe em sua totalidade, e funciona bem como um *container* para outras técnicas, metodologias e práticas. Dessa forma, é uma metodologia que pode ser aplicada em outros processos.

No projeto, serão realizados os seguintes eventos: *Sprint*, reunião de planejamento da *Sprint* e revisão da *Sprint*. Também serão utilizados o *Backlog* do Produto e o *Backlog* do *Sprint*, como artefatos do projeto.

3.5. TDD

Muitas das práticas sugeridas por métodos ágeis têm o objetivo de aumentar a quantidade e qualidade de *feedback* por parte da equipe do projeto em relação ao cliente e da qualidade (interna e externa) do código produzido pela equipe (ANICHE, 2012).

Segundo Aniche (2012), o Desenvolvimento Guiado por Testes, conhecido como TDD ou *Test-Driven Development*, popularizado por Kent Beck, é um das práticas ágeis no qual o foco é dar *feedback*. Baseia-se na repetição das atividades: escrita de um teste que falha, implementação de funcionalidade que faz o teste passar e refatoração do código para remover duplicação de código ou dados gerados pelo processo. A Figura 11 representa estas atividades.

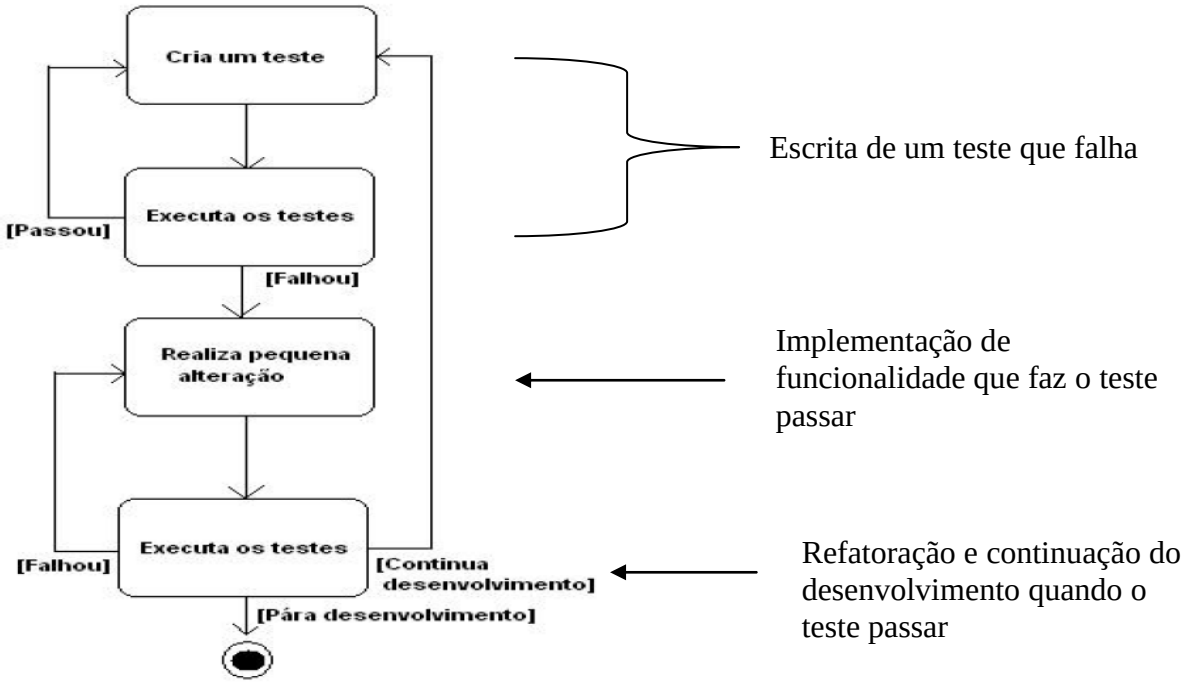

Figura 11 – Atividades do TDD

O TDD é uma técnica de programação onde o principal objetivo é escrever código funcional a partir de um teste que tenha falhado. O efeito deste processo é a obtenção de um código fonte bem testado (BAUMEISTER, 2002 apud GASPARETO, s/d).

O TDD tem como consequência uma bateria de testes de unidade. Esta prática contribui com a diminuição de erros de regressão, que ocorrem quando a implementação de uma nova funcionalidade quebra outra já existente além de prover segurança durante os processos de refatoração de código feitas durante o desenvolvimento do software. Além disso, a cobertura de código pelos testes também tende a ser alta, já que o teste será sempre escrito antes de se implementar uma nova funcionalidade (ANICHE, 2012).

No projeto, o TDD será utilizado antes da implementação dos métodos, visando maior cobertura de código. Também será utilizado, essencialmente, para testar os métodos, de forma a garantir que as combinações de valores expostas no LIE sejam testadas.

3.6. TOAD

O TOAD™ Data Point é uma ferramenta utilizada para consulta multiplataforma e integração de dados. Permite a conexão com fontes de dados relacionais tradicionais, como Oracle, SQL Server, Teradata e com fontes que não pertencem a banco de dados, como Excel.

Foi produzido pela DELL Software e possibilita consulta a diversas bases de dados por meio de *Query Buider*, no qual, por meio de interface visual o usuário constrói sua consulta e por meio de código direto, no qual o usuário digita o código SQL da consulta. Além disso, o TOAD permite que o usuário entenda a estrutura do banco de dados, suas tabelas e relacionamentos além de prover uma funcionalidade para automatização de consultas, onde se pode escolher data e horário para uma determinada consulta ser executada.

3.7. SQL Developer

O Oracle SQL Developer é uma ferramenta desenvolvida pela Oracle Corporation, e é disponibilizada para profissionais desenvolvedores e administradores de banco de dados Oracle. Também pode ser usado por usuários que necessitam usar os dados dos bancos de dados Oracle para alguma finalidade (JUNIOR, 2011).

A ferramenta acessa bancos de dados da Oracle a partir da versão 9.2.0.1, e pode ser executado em sistemas operacionais Windows, Linus e Mac OS X. Segundo Junior (2011) entre suas funcionalidades, destacam-se:

- Prover visão geral do esquema de um banco de dados Oracle;
- Executar instruções SQL, sejam elas DDL, DMS, DCL ou DTC e imprimir seus resultados;
- Criar e executar *scripts* SQL e PL/SQL; e
- Conceder ou revogar privilégios a usuários.

3.8. Eclipse

O Eclipse é uma IDE, *open source*, cuja principal característica é o desenvolvimento baseado em *plug-ins*. Reúne diversas ferramentas de apoio ao desenvolvimento de software, tais como:

- *Editor* – Edita o código-fonte do programa;
- *Compiler* – Compila o código-fonte, transformando-o em código de máquina;
- *Debugger* – Localiza erros;
- Geração de Código – Gera códigos que são usados com frequência; e
- *Automated tests* – Realiza testes automatizados no software.

A plataforma foi desenvolvida para construir ambientes de desenvolvimento ingrados e ferramentas arbitrárias. Suporta ferramentas que manipulam HTML, Java, C, JSP, EJB, XML e GIF (Eclipse.org, 2006).

3.9. EclEmma

Cobertura de código consiste na porcentagem dos requisitos que foram testados '*versus*' o total de requisitos gerados (COPELAND, 2004 apud RINCON, 2011). Esse percentual irá indicar quanto do código dos métodos de um projeto estão cobertos por testes. Esta métrica, associada ao desenvolvimento orientado a testes, fornece uma maior abrangência do que está sendo desenvolvido, tornando a implementação coesa com relação aos requisitos do sistema (BECK, 2004 apud ELIAS; WILDT, 2008).

Para medição dessa métrica, existe a ferramenta EclEmma, que serve para medir a cobertura de código java. Ela mostra a análise de cobertura diretamente no código no Eclipse. Seu funcionamento está diretamente ligado às execuções dos testes do *JUnit*. Os resultados da cobertura são destacados nos editores de código-fonte java (EclEMMA.org, s/d).

4. Proposta

Esta seção apresenta a descrição da proposta deste TCC e o processo que foi seguido para implementação da camada de modelo do sistema Educacenso.

4.1. Descrição da Proposta

Como informado anteriormente, o objetivo deste trabalho foi de desenvolver uma biblioteca em que as regras do sistema Educacenso estejam implementadas. Esta biblioteca servirá como camada de modelo para outros sistemas de gestão escolar.

Para alcançar este objetivo, foram realizadas as seguintes atividades expostas na Tabela 4:

Tabela 4 – Atividades para alcance do objetivo geral

ID	Atividade	Descrição
A1.	Engenharia reversa do sistema Educacenso.	Entender o funcionamento do sistema Educacenso, por meio do estudo dos fluxos da ferramenta e de sua documentação (foco no LIE).
A2.	Engenharia reversa do banco de dados do sistema Educacenso.	Entender a estrutura do banco de dados do Sistema Educacenso, buscando compreender quais são as principais tabelas e seus relacionamentos, chaves primárias e secundárias, dentre outros aspectos.
A3.	Geração dos scripts de criação de tabelas.	Gerar os scripts de criação das tabelas do banco de dados do sistema Educacenso, por meio do uso da ferramenta SQL Developer, com o intuito de replicar o banco de dados.
A4.	Criar o banco de dados.	Fazendo uso dos scripts da atividade anterior, será criado um novo banco de dados, o qual será utilizado na modelagem objeto relacional prevista para geração de classes java.

A5.	Gerar classes java a partir do banco de dados replicado.	Fazendo uso do banco de dados replicado e da ferramenta Eclipse, serão geradas as classes java.
A6.	Projetar a arquitetura do software.	Projetar a arquitetura do software a ser implementado, com foco na camada de modelo, onde serão implementadas as regras de negócio do sistema, que estão expostas no LIE.
A7.	Implementar as regras.	Implementar as regras expostas no LIE, utilizando a metodologia de desenvolvimento Scrum, com base no processo definido.

Para fins de melhor entendimento dos resultados deste projeto, dividiu-se as atividades expostas na Tabela 4 em duas fases: Fase A, composta pelas atividades A1 a A6, e Fase B, composta pela atividade A7.

Para a realização das atividades A4 a A7, foi definido um processo usando a "*Structured Analysis and Design Technique*" (SADT), proposto por ROSS (1985). A Figura 12 representa este processo.

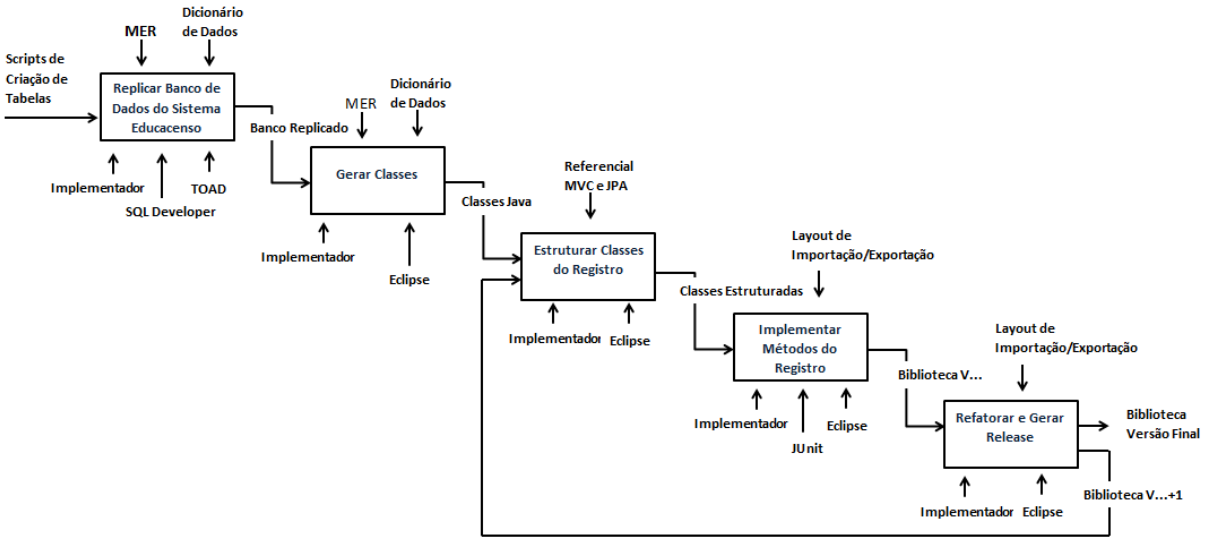

Figura 12 – Processo para execução do projeto

Como visto na Figura 12, este processo contém as seguintes atividades:

- Replicar o banco de dados do sistema Educacenso – após realizar a engenharia reversa do sistema Educacenso e do seu banco de dados, bem como gerar os scripts de criação das tabelas, será feita a replicação do banco de dados. Para isto, serão necessárias as ferramentas SQL *Developer* para geração dos Scripts de criação das tabelas e do TOAD para execução dos scripts. Para controle dessas atividades, serão utilizados o dicionário de dados do banco e o modelo de entidade e relacionamento. Ao final se terá um banco de dados com a mesma estrutura de tabelas e relacionamentos do banco de dados do sistema Educacenso;

- Gerar classes – usando o banco de dados criado e as ferramentas do JPA, serão geradas as classes java do projeto. Neste momento será utilizado o TOAD para fornecer os dados de conexão com o banco de dados (servidor, SID e outros) e o Eclipse que possui a funcionalidade JPA *tools,* que torna possível a geração de classes java a partir de tabelas de um banco de dados;

- Estruturar classes do registro – estruturar as classes que estão envolvidas em um registro do LIE para que atendam a arquitetura definida. Neste momento, serão utilizados o referencial teórico de JPA e de MVC para que as classes necessárias a um registro sejam estruturadas de forma a atenderem a arquitetura definida. Também será utilizado o Eclipse como IDE de implementação da camada de modelo;

- Implementar métodos do registro – serão implementados os métodos que atendem as regras expostas no LIE. Para isto, será utilizado o LIE como principal documentação das regras a serem implementados e o Eclipse; e

- Gerar *release* – Gerar nova release do projeto. Neste momento, se fará uso do Eclipse.

A Tabela 5 expõe as atividades do processo modelado em SADT com seus respectivos controles e mecanismos.

Tabela 5 – Atividades, controles e mecanismos

Atividade	Controles	Mecanismos
Replicar o banco de dados do sistema Educacenso	MER – Modelo de Entidade e Relacionamento que será	SQL Developer – Usado para gerar os *scripts* de criação

	utilizado para controlar a criação das tabelas e dos seus relacionamentos; e Dicionário de Dados – Utilizado para controlar a criação das tabelas e seus relacionamentos.	das tabelas; e TOAD – Usado para execução dos *scripts* de criação das tabelas.
Gerar classes	MER – Utilizado para verificar se o código gerado atende aos relacionamentos entre as tabelas; e Dicionário de Dados – Utilizado para verificar se os tipos de dados das classes atendem aos tipos de dados das tabelas.	Eclipse – Utilizado por fornecer a ferramenta JPA de gerar classes a partir das tabelas do banco de dados.
Estruturar classes do registro	Referencial MVC e JPA – Utilizado para auxiliar na estruturação das classes de forma que atendam ao padrão MVC e a especificação JPA.	Eclipse – Utilizada para o desenvolvimento do código da camada de modelo.
Implementar métodos do registro	Layout de Importação/Exportação – Utilizado para indicar as regras que devem ser implementadas.	Eclipse – Utilizada para o desenvolvimento do código da camada de modelo. e JUnit – Utilizado por fornecer um conjunto de classes necessárias à construção dos testes unitários.

Refatorar e gerar *release*	Layout de Importação/Exportação – Utilizado para garantir que após a refatoração, não foram prejudicadas.	Eclipse – Utilizada para o desenvolvimento do código da camada de modelo.

A Tabela 6 traz de forma estruturada as atividades do processo modelado em SADT e seus artefatos de entrada e saída.

Tabela 6 – Atividades e artefatos de entrada e de saída

Atividade	Artefatos de Entrada	Artefatos de Saída
Replicar o banco de dados do sistema Educacenso	Scripts de criação das tabelas do banco de dados do sistema Educacenso.	Banco de dados semelhante ao utilizado pelo sistema Educacenso em termos de estrutura e relacionamento entre tabelas.
Gerar classes	Banco de dados replicado.	Classes java
Estruturar classes do registro	Classes java	Classes java utilizadas pelo registro mapeadas no arquivo *persistence.xml* e com os tipos de seus atributos atualizados.
Implementar métodos do registro	Release mais atual do projeto.	Código com métodos de verificação implementados.
Refatorar e gerar *release*	Código com métodos de verificação implementados.	Nova release do projeto.

Como se pode verificar, as três últimas atividades representadas na Figura 12 e descritas na Tabela 6 serão repetidas para cada registro do LIE, ocorrendo uma

retroalimentação como forma de se evoluir a camada de modelo. Este sistema de *feedback* faz uso da 8ª Lei de Lehman, que estabelece que os processos de evolução incorporam sistemas de *feedback* com vários agentes e *loops* (LEHMAN, 2001 apud RIBEIRO, s/d).

Considerando a retroalimentação, as três últimas atividades descritas na Tabela 6 serão realizadas em um processo iterativo e incremental, sendo que cada iteração do processo será responsável pela implementação das regras de um registro do LIE.

4.2. Metodologia de Desenvolvimento

Conforme visto na Seção 4.1 as três últimas atividades descritas na Tabela 6 serão desenvolvidas em um processo iterativo e incremental, visto que a construção da camada de modelo será feita registro a registro, evoluindo a camada e requisitando a repetição das atividades de estruturação das classes, implementação dos métodos e refatoração do código.

Para a execução das três últimas atividades expostas no processo apresentado pela Figura 12, será utilizado o Scrum, por definir um processo iterativo e incremental, considera-se que o *framework* atende ao estabelecido pelo processo.

O *Product Backlog*, lista de funcionalidades a serem implementadas em um projeto, compreenderá o próprio LIE. As *Sprints* serão divididas para a implementação das regras de cada registro apresentado no layout. Dessa forma, os *Sprints Backlogs* serão os próprios registros do layout. A Figura 13 representa este processo:

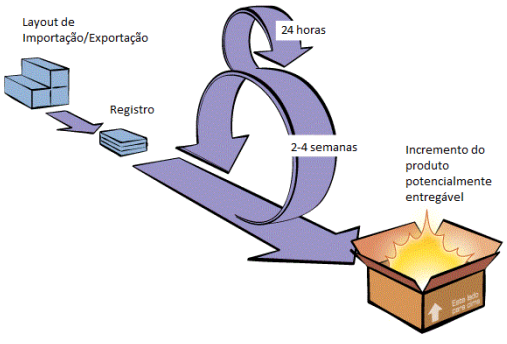

Figura 13 – Metodologia do Scrum para implementação da camada de modelo

Conforme mostrado na Figura 13, cada *Sprint* levará entre 2 a 4 semanas para ser concluída. Todos os dias será feito um planejamento do que será desenvolvido. Este planejamento é representado pelas 24 horas na Figura 13.

4.3. Arquitetura

Para implementação deste projeto, projetou-se utilizar o padrão MVC como padrão arquitetural, devido aos diversos benefícios apresentados na Seção 3.3 deste TCC. Dado o objetivo de implementar uma camada de modelo do sistema Educacenso, o foco deste projeto não está em designar a forma de implementação das outras camadas do MVC. Porém, ao se projetar a arquitetura do projeto, fez-se uma estrutura preparada para implementação de determinados *frameworks* do padrão MVC e outras tecnologias para a camada de controle e de visualização.

A estrutura do projeto java foi projetada segundo a Figura 14:

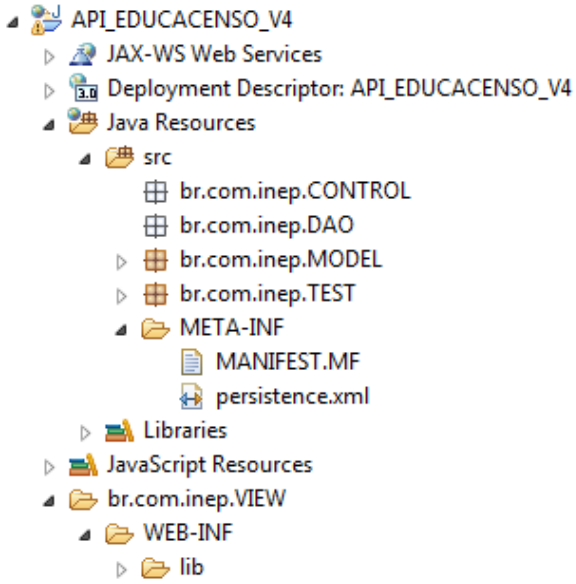

Figura 14 – Estrutura do projeto java

O pacote "br.com.inep.CONTROLE" será responsável por abrigar a camada de controle do projeto. Para esta camada, foi projetado o uso do *framework* do MVC "Struts 2", um controlador baseado no *WebWork* que é responsável por fazer um direcionamento de acordo com requisições. Neste *framework* está configurado qual *action* deve ser chamada para cada caso (CAELUM, s/d).

O pacote "br.com.inep.MODEL" é responsável por abrigar a camada de modelo do sistema, onde serão implementados os métodos de verificação expostos no LEI. Já o pacote "br.com.inep.DAO" será responsável por abrigar as classes que fazem operações sobre o banco de dados, isto é, consultas, inserções, alterações e deleções, de acordo com as funcionalidades do sistema.

A pasta “br.com.inep.VIEW” abrigará arquivos .jsp, tecnologia recomendada para a implementação da camada de visualização do projeto. Na subpasta “lib” serão armazenados os arquivos .jars necessários para o funcionamento do *Hibernate* e do *framework* Struts 2.

Com esta estrutura, o projeto poderá responder as especificações do padrão MVC, que com as tecnologias definidas, ficará conforme representa a Figura 15. Os métodos para verificação dos dados do Censo Escolar serão implementados sobre a camada de modelo, representada na Figura 15.

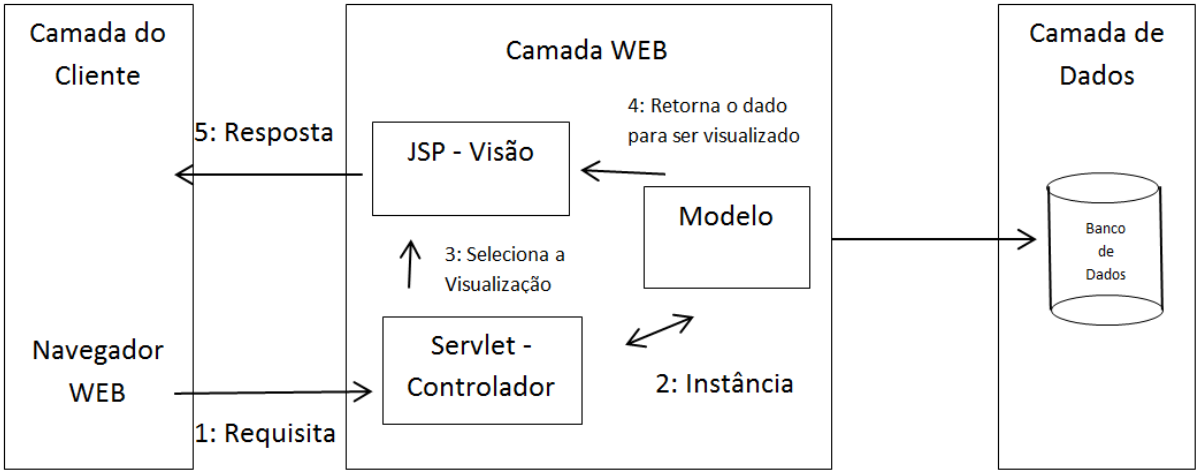

Figura 15 – Arquitetura do Projeto

5. Resultados da Fase A

Para execução da Fase A deste trabalho, foram seguidas as atividades descritas na Tabela 4. Primeiramente, foi feita uma engenharia reversa do banco de dados do sistema Educacenso, a qual será apresentada na Seção 5.1. Após o estudo do banco de dados, foram gerados os *scripts* de criação das tabelas para criação de um novo banco de dados para uso deste TCC. Também foi feita uma engenharia reversa do Educacenso, buscando descrever o fluxo principal e sequencias das principais funções do sistema. Os resultados desta engenharia reversa do sistema Educacenso se encontram na Seção 5.2. Depois disso, foram geradas as classes do projeto, de acordo com a demanda do registro do LIE cujos resultados estão expostos na Seção 5.3.

5.1. Engenharia reversa do banco de dados do sistema Educacenso

Com a finalidade de entender a estrutura do banco de dados do sistema Educacenso, foi feita uma análise usando se o modelo de entidade e relacionamento. O banco de dados possui um total de 82 tabelas e uma *view* para guardar os dados de usuários. Com base no MER do banco, primeiramente, foi feito um estudo de cada tabela a fim de poder descrever sua função e seus principais dados armazenados. O resultado desta primeira atividade se encontra no Anexo 1. Após ter sido feita a descrição de cada tabela, foi construída a Tabela 7 a qual traz as principais tabelas do banco de dados, considerando os registros do LIE, juntamente com suas chaves primárias e secundárias e tabelas de relacionamento:

Tabela 7 – Principais tabelas e relacionamentos

Tabela	Chave Primária	Chaves Secundárias	Tabela das Chaves Secundárias
TAB_ENTIDADE	PK_COD_ENTIDADE	FK_ANO_CENSO	TAB_ANO_CENSO
		FK_COD_MUNICIPIO	TAB_MUNICIPIO
		FK_COD_ESTADO	TAB_ESTADO
		FK_COD_DISTRITO	TAB_DISTRITO
TAB_TURMA	PK_COD_TURMA;	FK_ANO_CENSO	TAB_ANO_CENSO

	FK_ANO_CENSO; E FK_COD_ENTIDADE	FK_COD_ENTIDADE	TAB_ENTIDADE
		FK_COD_MOD_ENSINO	TAB_MOD_ETAPA_ENS
		FK_COD_ETAPA_ENSINO	TAB_MOD_ETAPA_ENS
		FK_COD_CURSO_PROF	TAB_CURSOS_ED_PROF
		FK_COD_TIPO_TURMA	TAB_TIPO_TURMA
TAB_DOCENTE	PK_COD_DOCENTE	FK_COD_MUNICIPIO_DEND	TAB_MUNICIPIO
		FK_COD_ESTADO_DNASC	TAB_ESTADO
		FK_COD_PAIS_ORIGEM	TAB_PAIS
		FK_COD_MUNICIPIO_DNASC	TAB_MUNICIPIO
		FK_COD_ESCOLARIDADE	TAB_ESCOLARIDADE
		FK_COD_ESTADO_DEND	TAB_ESTADO
TAB_ALUNO	PK_COD_ALUNO	FK_COD_MUNICIPIO_NASC	TAB_MUNICIPIO
		FK_COD_MUNICIPIO_END	TAB_MUNICIPIO
		FK_COD_ORGAO_EMISSOR	TAB_ORGAO_EMISSOR
		FK_COD_ESTADO_RG	TAB_ESTADO
		FK_COD_ESTADO_CERTIDAO_CIVIL	TAB_ESTADO
		FK_COD_PAIS_ORIGEM	TAB_PAIS
		ID_CARTORIO	TC_CARTORIO

Para construção da Tabela 7, foi utilizado o MER do banco de dados do Sistema Educacenso, o qual é apresentado pela Figura 16. No MER apresentado pela Figura 16, só são apresentadas as chaves primárias e estrangeiras, que são relevantes para se entender os relacionamentos entre as tabelas.

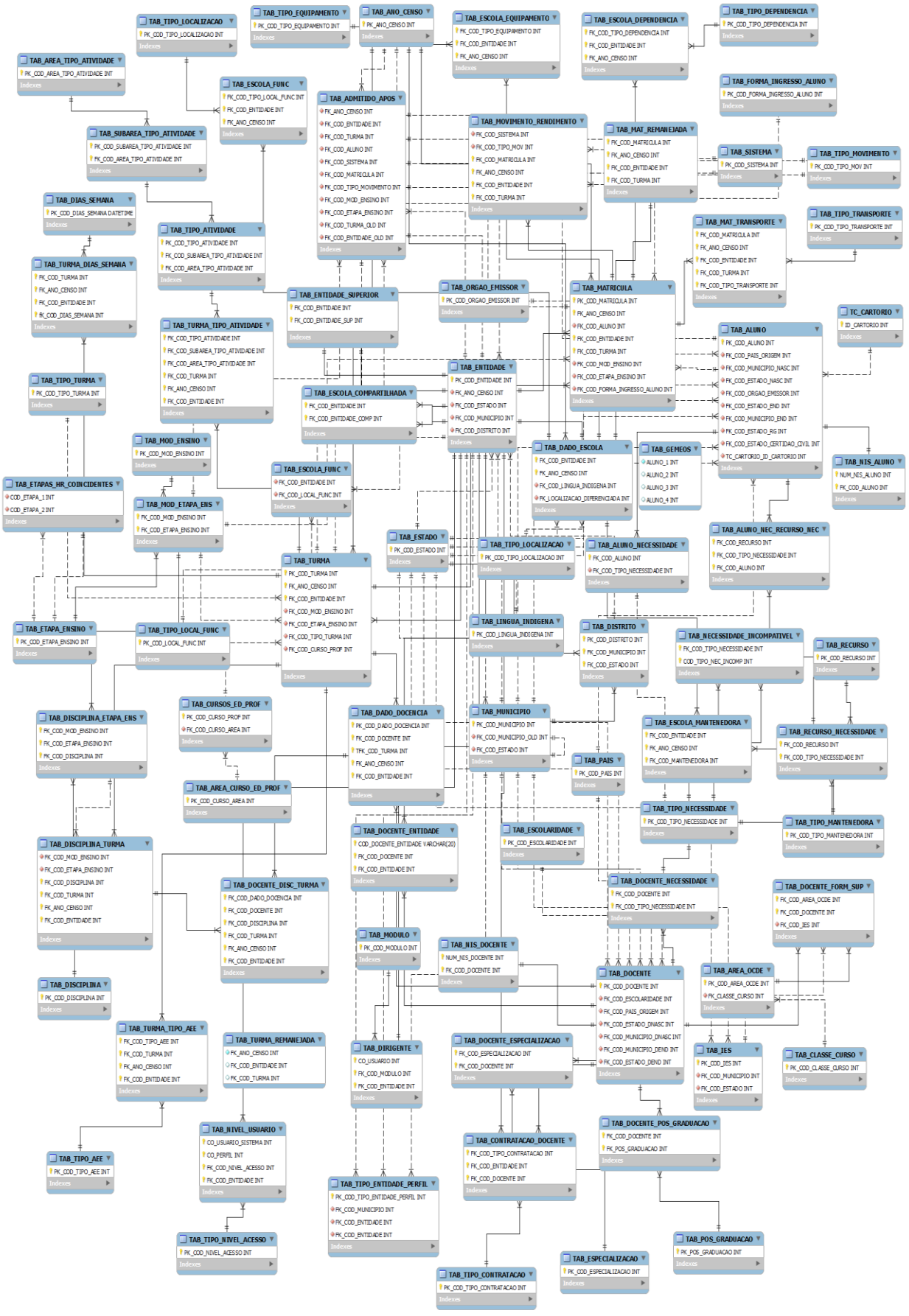

Figura 16 – MER do banco de dados do sistema Educacenso

5.2. Engenharia Reversa do Sistema Educacenso

O fluxo principal para se responder o Censo Escolar no sistema Educacenso compreende o exposto pela Figura 17:

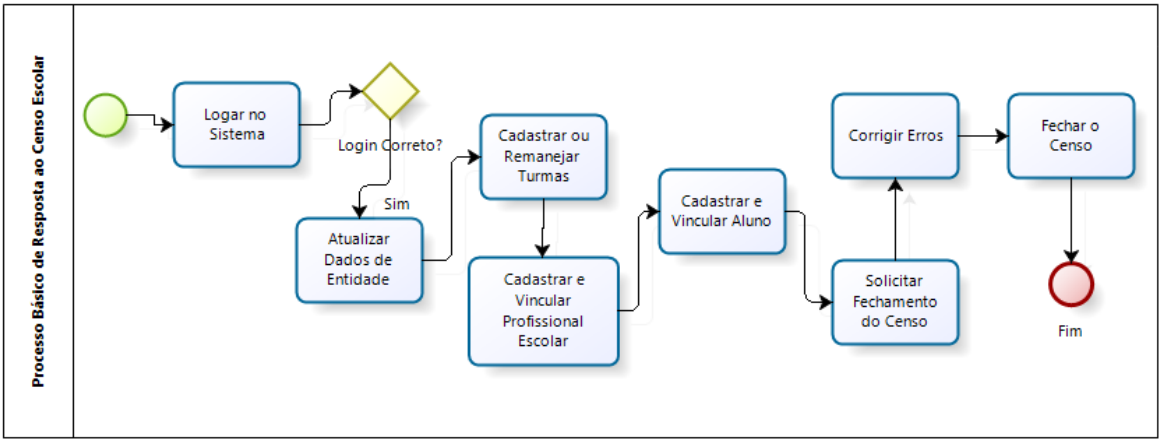

Figura 17 – Fluxo principal para se responder o Censo Escolar

Conforme mostrado na Figura 17, primeiramente o gestor escolar faz ***login*** no sistema. Após fazer o ***login***, o usuário terá que **atualizar os dados da escola** a qual ele é responsável. Após **atualizar os dados**, ele **cadastrará ou remanejará turmas**. No caso de remanejamento de turmas, todos os dados de uma turma do ano anterior, inclusive os vínculos com docentes e alunos serão mantidos para o ano atual.

Após o **cadastro de turma**, ele terá de **cadastrar docentes** e logo após **vincula-los às turmas**, juntamente com suas respectivas disciplinas. Depois, o usuário irá **cadastrar alunos** e em seguida, **vincula-los às turmas**. Ao final, ele fará a **solicitação de fechamento do Censo**. Neste momento o sistema Educacenso irá mostrar os possíveis erros cometidos durante o cadastro e os erros cruzados para **correção**. Após a **correção desses erros**, o gestor irá **fechar o Censo**.

O diagrama de sequência do sistema (DSS) exposto pela Figura 18 mostra com maiores detalhes a interação do usuário com o sistema Educacenso:

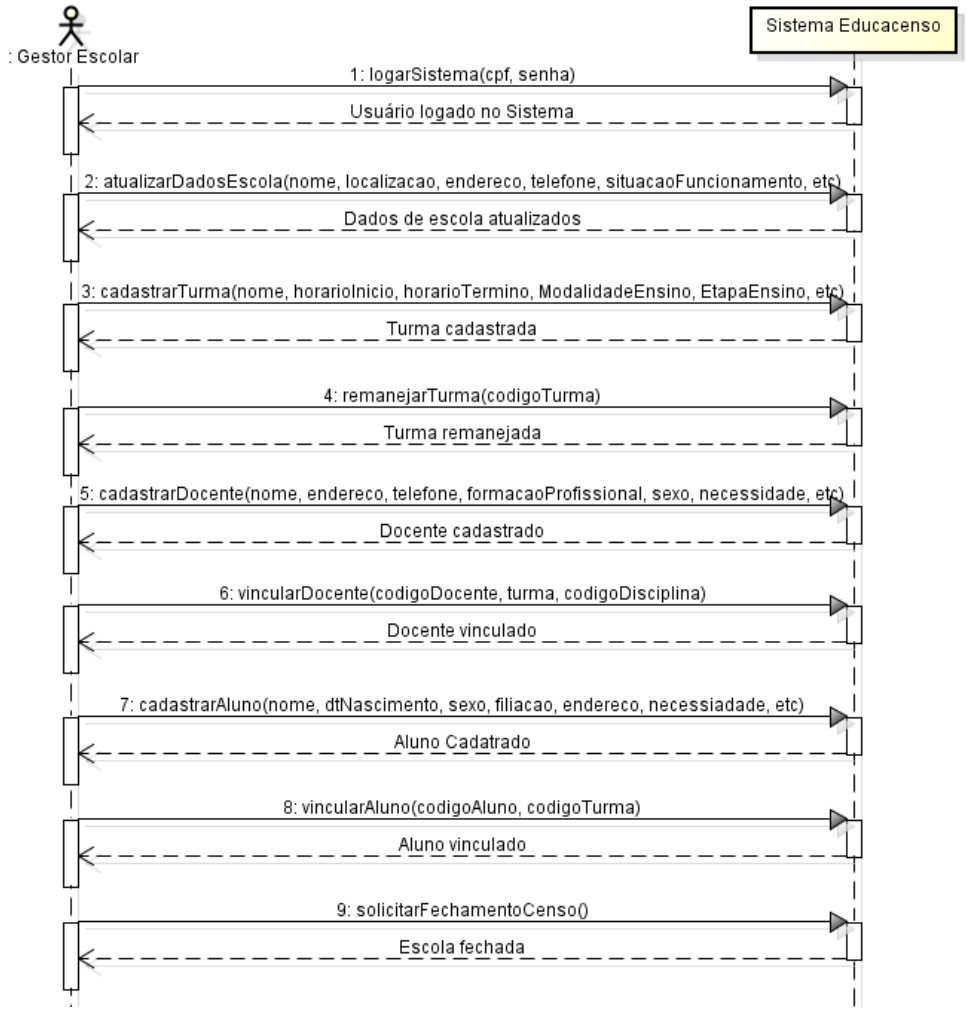

Figura 18 – DSS do sistema Educacenso

5.3. Classes para os Registros

As duas primeiras atividades do processo descrito pela Figura 12 foram executadas no projeto, bem como as atividades 3 e 4 para todos os registros do LIE.

A primeira atividade, descrita pelo processo exposto pela Figura 12, “Replicar Banco de Dados do Sistema Educacenso” teve como resultado um banco de dados, semelhante ao usado pelo Educacenso, em termos de estrutura das tabelas, *constraints* e relacionamentos. No total, foram criadas 82 tabelas, as quais estão descritas no Anexo 1.

A segunda atividade, também descrita pelo processo exposto pela Figura 12, “Gerar Classes” teve como resultado as classes java. Para realização desta atividade, foi necessário que as seguintes tarefas descritas na Tabela 8 fossem executadas.

Tabela 8 – Atividades para geração de classes java

Tarefa	Descrição
Criar Projeto JPA	Criação no Eclipse de um projeto JPA, onde foram armazenadas as classes java geradas.
Criar Conexão com o Banco de Dados Replicado	Aciona-se a ferramenta JPA *Tools* e em seguida, escolhe-se a opção *Generate Entities from Tables.* Adiciona-se uma nova conexão Oracle e entra-se com os dados da conexão (SID, *Host, Port number*, *User name e Password*).
Selecionar Tabelas do Banco de Dados	Após criar a conexão com o banco de dados, escolhe-se o *Schema* onde se encontram as tabelas. Depois, selecionam-se todas as tabelas do banco de dados replicado.

Após a execução dessas atividades, obteve-se 95 classes java, as quais estavam com diversas *Annotations*, de acordo com as colunas e com os relacionamentos das tabelas do banco de dados replicado, mas ainda não estavam mapeadas.

Com a finalidade de se levantar quais as classes java seriam necessárias para implementação das regras de cada registro do LIE e tendo-se o objetivo de estabelecer um controle da situação do projeto ao longo do seu ciclo de vida, elaborou-se a Tabela 9, que contém os registros, número de campos, número de tabelas envolvidas, número de classes java envolvidas (com base nas classes geradas) e número de regras gerais (fixo, tamanho, formato e obrigatório) e específicas de cada registro.

Tabela 9 – Tabelas, classes e regras de cada registro

Registro	Número de Campos	Tabelas Envolvidas	Número de tabelas envolvidas	Número de classes envolvidas	Número de Regras Gerais	Número de Regras Específicas
00	35	tab_entidade; tab_dado_escola; tab_escola_mantenedora; tab_estado;e tab_municipio;	5	3	140	78

10	130	tab_dirigente; tab_escola_func; tab_dado_escola; tab_escola_compartilhada; tab_escola_dependencia; tab_escola_equipamento; tab_tipo_dependencia; tab_tipo_equipamento; vw_usuario.	9	8	520	100
20	65	tab_turma; tab_turma_dias_semana; tab_dias_semana; tab_turma_tipo_atividade; tab_turma_tipo_aee; tab_disciplina_turma; tab_disciplina; tab_area_curso_ed_prof.	8	15	260	61
30	24	tab_docente; tab_docente_necessidade; tab_estado; tab_municipio; tab_pais.	5	3	96	64
40	13	tab_docente.	1	1	52	31
50	46	tab_docente; tab_docente_form_sup; tab_docente_pos_graduacao; tab_docente_especializacao.	4	8	184	113
51	21	tab_dado_docencia; tab_docente_disc_turma.	2	3	84	18
60	40	tab_aluno; tab_aluno_necessidade; tab_aluno_nec_recurso; tab_municipio; tab_estado; tab_pais.	6	5	160	62
70	31	tab_aluno.	1	3	124	99
80	24	tab_matricula; tab_mat_transporte.	2	3	96	32
TOTAL:	**429**	-	**41**	**57**	**1716**	**658**

Após as classes terem sido geradas, foi criado um novo projeto, onde foi implementada a arquitetura descrita na Seção 4.2. Antes de executar a atividade “Estruturar Classes do Registro”, foi necessário configurar o JPA com o *Hibernate*. Para isso, foram incluídos alguns arquivos de bibliotecas (JARs) do *Hibernate ORM* 4.2.7.

Também foi feito o mapeamento das classes em um arquivo chamado persistence.xml, necessário para o correto funcionamento do projeto. Este arquivo também fornece os dados para conexão com o banco de dados.

Para implementação dos métodos de verificação, que visam à cobertura das regras do LIE, foi utilizado o TDD. Foi criado um pacote onde as classes de testes foram armazenadas. Para cada regra disposta no LIE, foi feito primeiramente um teste na classe específica. Após o teste ser criado, o método foi criado na respectiva classe. Os resultados desta fase estão expostos na Seção 6.

6. Resultados da Fase B

Esta Seção irá apresentar os resultados da Fase B deste projeto, a qual trata da implementação dos métodos dos registros expostos no LIE. As Seções 6.1 a 6.10 vão apresentar a execução de cada *Sprint* do projeto, ou seja, os resultados da implementação de cada Registro do LIE, conforme definido na Seção 4.2. A Seção 6.11 vai apresentar os testes realizados sobre a camada de modelo implementada. Por fim, a Seção 6.12 vai apresentar como foi feito o gerenciamento do projeto.

6.1. *Sprint* 1 – Implementação dos Métodos do Registro 00

A Tabela 10 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 10 – *User Story* e tempo estimado para o Registro 00

User Story	*Story*: Criação dos métodos do Registro 00: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de escola: situação de funcionamento, data de início e término do ano letivo, nome da escola, latitude, longitude, endereço, telefone, e-mail, órgão regional, dependência administrativa, localização, categoria da escola privada, convênio com o poder público, mantenedoras, cnpj da escola privada e de sua mantenedora e regulamentação nos conselhos de educação.
Tempo Estimado	10 dias.

Para implementar os métodos deste registro, foram necessárias três classes java: Uma para descrever objetos do tipo entidade, outra para objetos do tipo dado escola e uma última para objetos de tipo mantenedora, para escolas que forem privadas. Primeiramente, as três classes que representam os tipos citados foram estruturadas no projeto. Após algumas correções de imperfeições produzidas no processo de geração de classes a partir do banco de dados, criou-se uma classe denominada “GeraTabelas”, que tem o objetivo de criar tabelas em um banco de dados com base nas *Annotations* de JPA para as classes correspondentes. Além de criar as tabelas, a execução desta classe permitiu alterações ou novas inserções de tabelas

no banco de dados, necessárias para a construção da camada de modelo. O uso do JPA permitiu assim que possíveis usuários da camada de modelo apenas tenham que criar um banco de dados no MySql, excluindo a necessidade de criação manual das tabelas.

Para criação dos métodos de verificação dos campos do Registro 00, foi utilizado o TDD. Após a criação do teste, foi criado o método. Para os campos numéricos, foram testados valores aceitáveis e valores seguintes à margem considerados não aceitáveis. Como exemplo, para o campo situação de funcionamento, que aceita valores entre 1 a 3, foram testados os valores 0, 1, 3 e 4. O mesmo método foi seguido para os outros campos numéricos. Os campos alfanuméricos foram testados com base no exposto nas colunas de regras e condicionalidades e validações do LIE. A Figura 19 traz o teste do campo situação de funcionamento funcionando. A Figura 20 traz o mesmo teste falhando quando é atribuído ao campo um valor inválido. A Figura 21 mostra e o método implementado para verificação da situação de funcionamento.

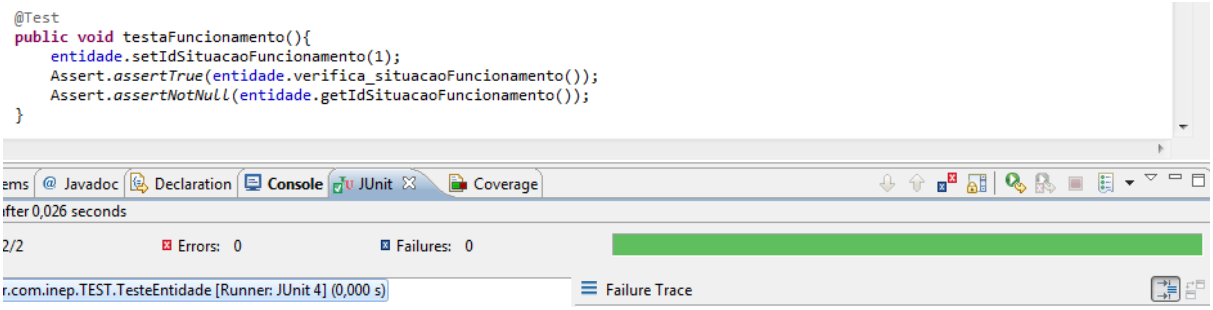

Figura 19 – Teste para verificação da situação de funcionamento

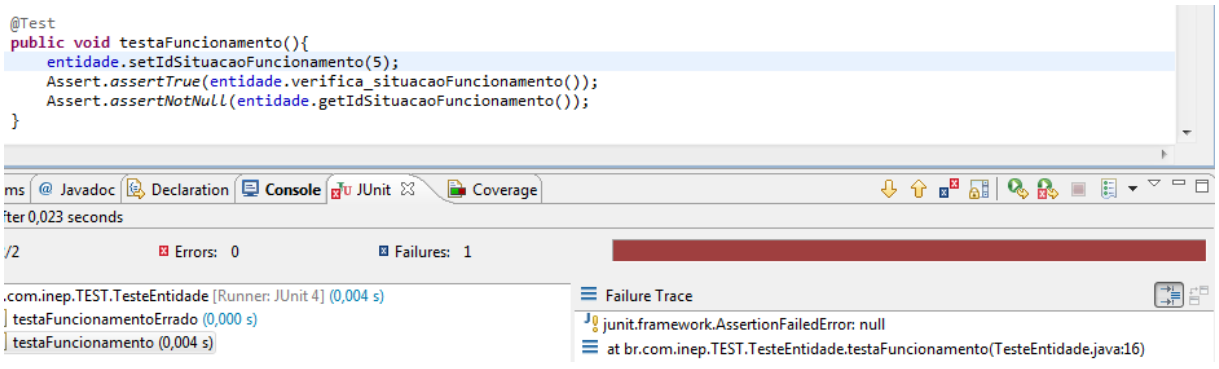

Figura 20 – Teste falhando ao atribuir valor inválido para situação de funcionamento

```
//REGISTRO 00 CAMPO 3 - VERIFICAÇÃO DA SITUAÇÃO DE FUNCIONAMENTO

    public boolean verifica_situacaoFuncionamento() {
        Pattern pattern = Pattern.compile("[a-zA-Z]");
        Matcher matcher = pattern.matcher(Integer.toString(this.idSituacaoFuncionamento));
        if(this.idSituacaoFuncionamento < 1  || this.idSituacaoFuncionamento > 4){
            return false;
        }else if(Integer.toString(this.idSituacaoFuncionamento).length() < 1 ||
                Integer.toString(this.idSituacaoFuncionamento).length() > 1 ||
                matcher.find()){
                    return false;
                } else
                    return true;
    }
```

Figura 21 – Método para verificação do campo situação de funcionamento

Conforme a Figura 22, alguns campos do LIE possuem nomes seguidos de um asterisco. Estes campos aceitam apenas valores contidos em chamadas Tabelas Auxiliares. Dessa forma, antes da criação dos métodos de verificação destes campos, foi feita uma carga desses valores em suas respectivas tabelas no banco de dados. No caso do Registro 00, os campos que estavam nesta situação eram: UF, município, distrito, DDD e órgão regional. Para estes campos, foi feito um método que verificava se o valor informado estava entre os valores contidos no banco de dados.

14	UF (*)	Sim	2	N	Sim	Deverá ser preenchido com o código do estado, de acordo com a "Tabela de UF".	O valor informado deve estar dentre os valores da "Tabela de UF".
15	Município (*)	Sim	7	N	Sim	Deverá ser preenchido com o código do município, de acordo com a "Tabela de Município".	O valor informado deve estar dentre os valores da "Tabela de Município" possíveis para o estado informado.
16	Distrito (*)	Não	2	N	Sim	Deverá ser preenchido com o código do distrito, de acordo com a "Tabela de Distritos".	O valor informado deve estar dentre os valores da "Tabela de Distrito" possíveis para o município e o estado informados.
17	DDD (*)	Sim	2	N	Não	Apenas números são aceitos.	O valor informado deve estar dentre os valores da "Tabela de Município" possíveis para o município informado.

Figura 22 – Campos do LIE com asterisco

Para o Registro 00, foram criados 31 casos de testes. Para verificação dos campos relativos à mantenedora da escola privada, apenas um método foi necessário. Com a finalidade de diminuir o acoplamento da classe Entidade, os métodos que buscavam informações no banco de dados foram transferidos a classes que ficaram no pacote br.com.inep.DAO.

Após a construção dos métodos, foi criada uma classe *Main* que têm a finalidade de testar o grupo de métodos de verificação dos campos do Registro 00. Caso o teste seja positivo, o objeto é persistido no banco de dados.

A Tabela 11 traz as classes entregáveis da *Sprint* 1 e o tempo para sua execução.

Tabela 11 – Entregáveis da *Sprint* 1

Classes na Camada de Modelo	Entidade.java; DadoEscola.java; TipoMantenedora.java; e GeraTabelas.java.
Classes de Teste	TesteEntidade.java; TesteDadoEscolaRegistro00.java; e TesteTipoMantenedora.java.
Classes DAO – para verificação junto ao Banco de Dados	MunicipioDAO.java; EstadoDAO.java; e OrgaoReginalDAO.java.
Número de Casos de Teste	31 casos de teste.
Tempo para Execução	6 dias.

Os seguintes casos de teste foram criados para verificar os campos relativos ao Registro 00:

1. testaFuncionamento();
2. testaPkCodigoEntidade();
3. testaDepAdm();
4. testaCep();
5. testaDDD();
6. verificaLocalizacao();
7. verificaNomeEscola();
8. verificaEndereco();
9. verificaEnderecoNumero();
10. verificaComplemento();
11. verificaBairro();
12. verificaUF();
13. verificaMunicipio();
14. verificaDistrito();
15. verificaTelefone();
16. verificaOutroTelefone();
17. verificaTelefonePublico1();
18. verificaFax();
19. verificaOrgaoRegional();
20. verificaLatitude();
21. verificaLongitude();
22. verificaEmail();

23. testaDataInicio();
24. testaDataTermino();
25. testaDatasInicioTermino();
26. testaCategoriaEscolaPrivada();
27. testaConvenioPoderPublico();
28. testaCnpj();
29. testaCnpjMantenedora();
30. testaRegulamentacao(); e
31. verificaMantenedora().

6.2. *Sprint* 2 – Implementação dos Métodos do Registro 10

A Tabela 12 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 12 – *User Story* e tempo estimado para o Registro 10

User Story	*Story*: Criação dos métodos do Registro 10: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de escola: gestor escolar (CPF, nome, cargo e endereço eletrônico), local de funcionamento, forma de ocupação do prédio escolar, compartilhamento do prédio com outra escola, código da escola a qual compartilha o prédio, água consumida pelos alunos, abastecimento de água, abastecimento de energia elétrica, esgoto sanitário, destinação do lixo, dependências existentes na escola, equipamentos existentes na escola, acesso à internet, total de funcionários, atividade complementar, AEE, modalidades de ensino ofertadas, etapas de ensino ofertadas, ensino fundamental organizado em ciclos, localização diferenciada da escola, materiais didáticos específicos para atendimento à diversidade sociocultural, educação indígena, espaço para turmas do Brasil Alfabetizado, abertura nos finais de semana e proposta pedagógica de formação por alternância.
Tempo Estimado	20 dias.

Para implementar os métodos do Registro 10, foram necessárias 8 classes java, para descrever objetos do tipo dado escola, dirigente, local de funcionamento, localização

diferenciada, escola compartilhada, dependências e equipamentos existentes na escola e língua indígena. Após estruturação das classes java no projeto, foi executada a classe GeraTabelas para atualizar o banco de dados, com a criação de novas tabelas onde os objetos citados são persistidos.

Conforme especificado pelo processo proposto para implementação da camada de modelo, foi utilizado o TDD. Primeiramente foram criados os testes e depois os métodos. Uma grande parte dos campos do Registro 10 aceitam apenas os valores 0 e 1. Para estes casos, foram testados valores entre -1 e 2 para validação dos métodos. A Figura 23 apresenta o teste criado para verificação do abastecimento de água. A Figura 24 apresenta o mesmo teste falhando ao atribuir valor inválido em um dos campos de abastecimento de água. A Figura 25 apresenta o método para verificação destes campos.

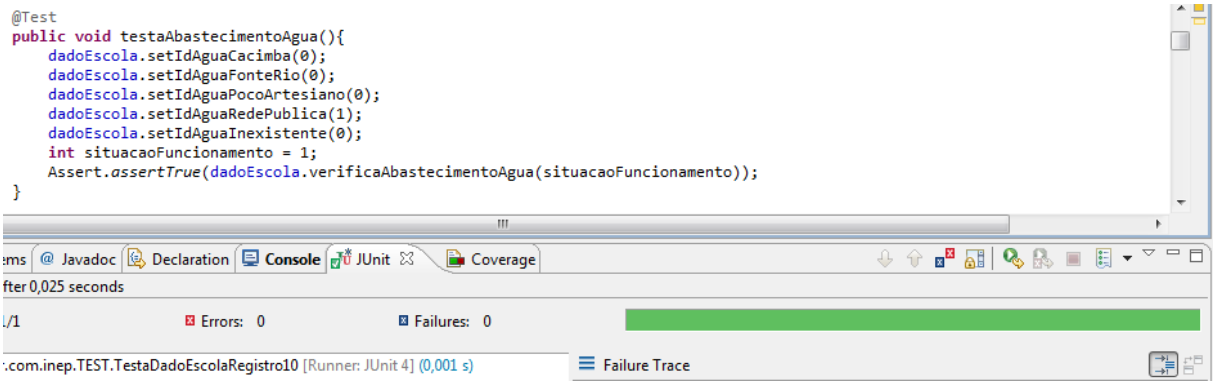

Figura 23 - Teste para verificação do abastecimento de água

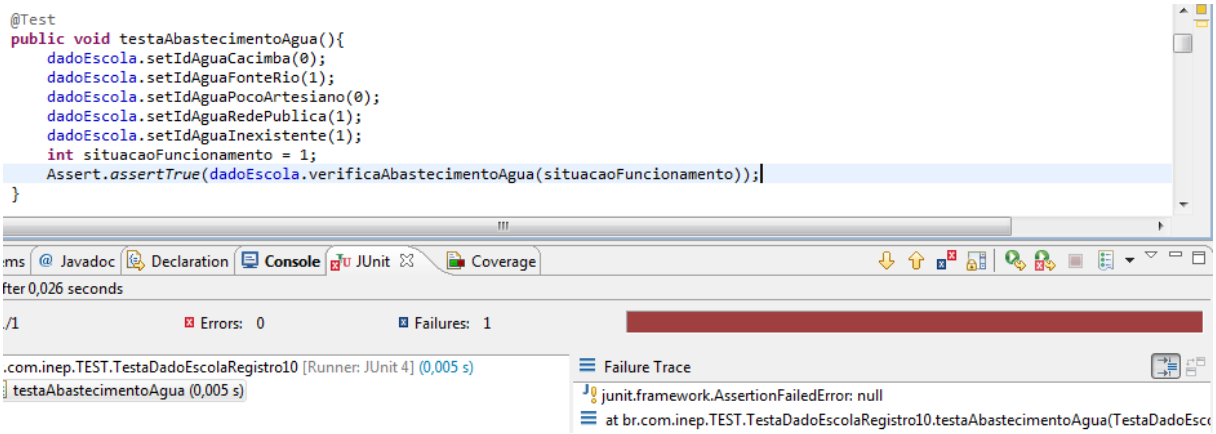

Figura 24 – Teste falhando ao atribuir valor inválido no abastecimento de água

```
//REGISTRO 10 CAMPOS 25 A 29 - VERIFICAÇÃO DO ABASTECIMENTO DE ÁGUA
    public boolean verificaAbastecimentoAgua(int situacaoFuncionamento) {
        if(verificaIdsNumericos(this.idAguaCacimba)
             || verificaIdsNumericos(this.idAguaFonteRio)
               || verificaIdsNumericos(this.idAguaPocoArtesiano)
                 || verificaIdsNumericos(this.idAguaRedePublica)
                   || verificaIdsNumericos(this.idAguaInexistente)
                     || this.idAguaCacimba < 0 || this.idAguaCacimba > 1
                       || this.idAguaFonteRio < 0 || this.idAguaFonteRio > 1
                         || this.idAguaPocoArtesiano < 0 || this.idAguaPocoArtesiano > 1
                           || this.idAguaRedePublica < 0 || this.idAguaRedePublica > 1
                             || this.idAguaInexistente < 0 || this.idAguaInexistente > 1
                               || (this.idAguaInexistente == 1 && (this.idAguaCacimba == 1
                                                                   || this.idAguaFonteRio == 1
                                                                   || this.idAguaPocoArtesiano == 1
                                                                   || this.idAguaRedePublica == 1))
                                  || situacaoFuncionamento != 1){
            return false;
        }
        return true;
    }

//VERIFICAÇÃO DE IDS NUMÉRICOS
    public boolean verificaIdsNumericos(int id){
        Pattern pattern = Pattern.compile("[a-zA-Z]");
        Matcher matcher = pattern.matcher(Integer.toString(id));
        if(matcher.find() || Integer.toString(id).length() != 1){
            return true;
        }else
            return false;
    }
```

Figura 25 – Método para verificação do abastecimento de água

Para o Registro 10 foram criados 46 casos de teste. Para os campos relativos ao código da escola compartilhada, foi necessário apenas um caso de teste, assim como para verificação do abastecimento de água, abastecimento de energia elétrica, esgoto sanitário, destinação do lixo, dependências e equipamentos existentes na escola e modalidade e etapas de ensino ofertadas. Para verificação do código da língua indígena, para escolas indígenas que ministram o ensino em alguma língua indígena, foi feito um método para verificação no banco da existência da língua informada.

A Tabela 13 traz os entregáveis da *Sprint* 2 e o tempo para sua execução.

Tabela 13 – Entregáveis da *Sprint* 2

Classes na Camada de Modelo	DadoEscola.java; Dirigente.java; TipoLocalFunc.java; TipoLocalizacao.java; EscolaFunc.java; EscolaCompartilhada.java; EscolaDependencia.java; EscolaEquipamento.java; e LinguaIndigena.java.
Classes de Teste	TestaDadoEscolaRegistro10.java; TesteDadoEscola.java; TesteDirigente.java; TesteEscolaCompartilhada.java;

	TesteEscolaDependencia.java; TesteEscolaEquipamento.java; e TesteEscolaFunc.java.
Classes DAO – para verificação junto ao Banco de Dados	LinguaIndigenaDAO.java.
Número de Casos de Teste	46 casos de teste.
Tempo para Execução	34 dias.

6.3. *Sprint* 3 – Implementação dos Métodos do Registro 20

A Tabela 14 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 14 – *User Story* e tempo estimado para o Registro 20

User Story	*Story*: Criação dos métodos do Registro 20: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de turma: horário de início e de término, dias da semana nos quais funciona, tipo de atendimento, turma participante do programa mais educação/ensino médio inovador, tipo de atividade complementar (caso turma seja de atividade complementar), atividades do atendimento educacional especializado (caso a turma seja de AEE), modalidade e etapa de ensino, código do curso de educação profissional (caso turma seja de etapa de educação profissional), disciplinas ministradas na turma e turma sem profissional escolar em sala de aula.
Tempo Estimado	15 dias.

Para implementar os métodos do Registro 20, foram necessárias 15 classes java para descrever objetos do tipo turma, dias da semana, tipo de turma, área de tipo de atividade, subárea de tipo de atividade, tipo de atividade, tipo de atividade da turma, tipo de AEE, modalidade de ensino, etapa de ensino, combinação de modalidade e etapa de ensino, área de

curso de educação profissional, curso de educação profissional, disciplina, disciplinas na turma e disciplina por etapa de ensino. Após estruturação dessas classes, foram geradas suas respectivas tabelas no banco de dados.

Para implementação dos métodos de verificação de cada dia da semana, foi necessário um único método. Dessa mesma forma foi feito para os campos relativos ao tipo de atividade para turmas de atividade complementar e AEE e para disciplinas na turma.

Para implementação dos métodos das atividades de turmas de atividade complementar e AEE e para implementação dos métodos dos campos de modalidade e etapa de ensino, foi necessário verificar se a escola informada para a turma está em funcionamento, com base no valor informado no Registro 00. Também foi necessário verificar se a escola oferece atividade complementar, AEE e a modalidade e etapa de ensino informada para a turma, com base no valor informado para a escola no Registro 10.

A Figura 26 mostra o caso de teste utilizado para testar o campo de modalidade de ensino. A Figura 27 mostra o método para verificação deste campo. A Figura 28 mostra o método que traz os dados da escola informada para a turma.

```
@Test
public void testaModalidade(){
    ModEtapaEn modalidade = new ModEtapaEn();
    ModEtapaEnPK modalidadePk = new ModEtapaEnPK();
    ModEtapaEn modalidadeInvalida = new ModEtapaEn();
    ModEtapaEnPK modalidadeInvalidaPk = new ModEtapaEnPK();
    modalidadePk.setFkCodModEnsino(1);
    modalidade.setId(modalidadePk);
    modalidadeInvalidaPk.setFkCodModEnsino(3); //MODALIDADE 3 (EDUCAÇÃO ESPECIAL) NÃO OFERECIDA PELA ESCOLA DE CÓDIGO 41602978
    modalidadeInvalida.setId(modalidadeInvalidaPk);
    turma.setModEtapaEn(modalidade);
    turma2.setModEtapaEn(modalidadeInvalida);
    int tipoTurma = 0;
    int codigoEscola = 41602978;
    Assert.assertTrue(turma.verificaModalidade(tipoTurma, codigoEscola));
    //ASSERT FALSE PARA INDICAR QUE O RETORNO DO MÉTODO É FALSE QUANDO INFORMADA TURMA COM MODALIDADE NÃO OFERECIDA PELA ESCOLA.
    Assert.assertFalse(turma2.verificaModalidade(tipoTurma, codigoEscola));
}
```

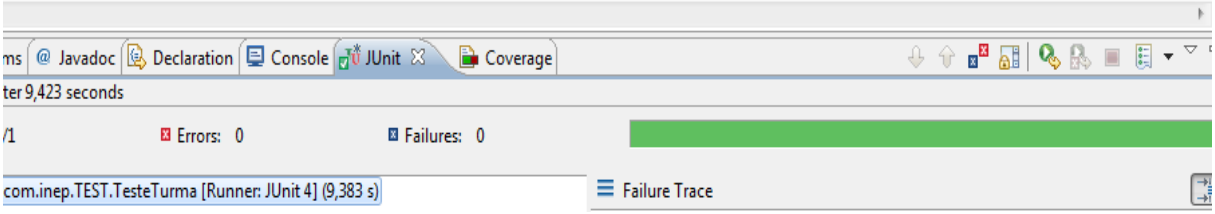

Figura 26 – Teste para verificação da modalidade de ensino

```java
//REGISTRO 20 CAMPO 36 - VERIFICAÇÃO DA MODALIDADE
    public boolean verificaModalidade(int tipoTurma2, int codigoEscola) {
        Pattern pattern =  Pattern.compile("[0-9]");
        Matcher matcher = pattern.matcher(Integer.toString(this.modEtapaEn.getId().getFkCodModEnsino()));
        DadoEscolaDAO dadoEscolaDAO = new DadoEscolaDAO();
        DadoEscola dadoEscola = dadoEscolaDAO.retornaDadosDadoEscola(codigoEscola);
        if(!matcher.matches()
          || Integer.toString(this.modEtapaEn.getId().getFkCodModEnsino()).length() != 1
            || this.modEtapaEn.getId().getFkCodModEnsino() < 1
              || this.modEtapaEn.getId().getFkCodModEnsino() > 3
                || tipoTurma2 == 4
                  || tipoTurma2 == 5
                    || (this.modEtapaEn.getId().getFkCodModEnsino() == 1 && dadoEscola.getIdModEnsinoRegular() != 1)
                      || (this.modEtapaEn.getId().getFkCodModEnsino() == 2 && dadoEscola.getIdModEnsinoEsp() != 1)
                        || (this.modEtapaEn.getId().getFkCodModEnsino() == 3 && dadoEscola.getIdModalidadeEja() != 1)){
            return false;
        }else
            return true;
    }
```

Figura 27 – Método para verificação da modalidade de ensino

```java
//MÉTODO PARA RETORNAR OS DADOS DA ESCOLA CONTIDOS NA TABELA DADO_ESCOLA
    public DadoEscola retornaDadosDadoEscola(int fkCodEntidade) {
        DadoEscola retorno = null;
        int codigoEntidade = fkCodEntidade;
        try {
            EntityManagerFactory factory = Persistence.createEntityManagerFactory("entidade");
            EntityManager manager = factory.createEntityManager();
            manager.clear();
            Query query = manager.createQuery("SELECT D FROM DadoEscola AS D " +
                    "WHERE D.id.fkCodEntidade = :paramCodigoEntidade");
            query.setParameter("paramCodigoEntidade", codigoEntidade);
            List<Object> encontrado = query.getResultList();
            manager.clear();
            manager.close();
            factory.close();
            for(Object dadoEscola : encontrado){
                DadoEscola dadoEscolaCast = (DadoEscola) dadoEscola;
                retorno = (DadoEscola) dadoEscola;
            }
        } catch (Exception e) {
            System.out.println("ESCOLA NÃO EXISTENTE NO BANCO DE DADOS.");
            System.out.println(e);
        }
        return retorno;
    }
```

Figura 28 – Método para retornar dados da escola

Para o Registro 20 foram criados 19 casos de teste. A Tabela 17 mostra os entregáveis dessa *Sprint* e o tempo para execução do seu processo de implementação.

Tabela 17 – Entregáveis da *Sprint* 3

Classes na Camada de Modelo	Turma.java; TurmaTipoAtividade.java; TipoTurma.java; TipoAtividade.java; TipoAee.java; ModEnsino.java; EtapaEnsino.java; ModEtapaEn.java; Disciplina.java; DisciplinaEtapaEn.java; DisciplinaTurma.java; DiasSemana.java; CursosEdProf.java; AreaCursoEdProf.java; SubareaTipoAtividade.java; e AreaTipoAtividade.java;

Classes de Teste	TesteTurma.java; e TesteTurmaTipoAtividade.java.
Classes DAO – para verificação junto ao Banco de Dados	DadoEscolaDAO.java; DiaSemanaDAO.java; ModEtapaEnsDAO.java; e TurmaTipoAtividadeDAO.java.
Número de Casos de Teste	19 casos de teste.
Tempo para Execução	25 dias.

6.4. *Sprint* 4 – Implementação dos Métodos do Registro 30

A Tabela 16 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 16 – *User Story* e tempo estimado para o Registro 30

User Story	*Story*: Criação dos métodos do Registro 30: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de docente: nome completo, e-mail, número de identificação social (NIS), data de nascimento, sexo, cor/raça, nome completo da mãe, nacionalidade, país de origem, estado e município de nascimento e profissional escolar com deficiência.
Tempo Estimado	15 dias.

Para implementação dos métodos do Registro 30, foram necessárias 3 classes java para descrever objetos do tipo docente, NIS do docente e tipo de deficiência. Após as classes terem sido estruturadas, foram geradas suas respectivas tabelas no banco de dados.

Para construção do método que verificasse o NIS, foi criada uma classe que, com base nas regras de validação do NIS, gerasse números válidos para este campo. Após isso, foi feito um método que verificasse se o NIS informado era válido. Da mesma forma foi feito para o campo de CPF. Os campos relativos a país de origem, estado e município de nascimento

foram verificados junto às respectivas tabelas auxiliares. Dois métodos foram criados para verificação das possíveis necessidades (deficiências) do docente: um para verificar se o código da necessidade é válido e outro para verificar a compatibilidade entre as necessidades informadas, caso o docente tenha mais de uma deficiência. A Figura 29 mostra o caso de teste para verificação do NIS. A Figura 30 traz o método para verificação deste campo.

```
@Test
public void testaNisDocente(){
    //Objeto com NIS válido
    nisDocentePK.setFkCodDocente("123654789567");
    nisDocentePK.setNumNisDocente("18108315730");
    nisDocente.setId(nisDocentePK);
    //Objeto com NIS inválido
    nisDocenteInvalidoPK.setFkCodDocente("123654789567");
    nisDocenteInvalidoPK.setNumNisDocente("27384765789");
    nisDocenteInvalido.setId(nisDocenteInvalidoPK);
    //Asserts
    Assert.assertTrue(nisDocente.verificaNumNisDocente());
    Assert.assertFalse(nisDocenteInvalido.verificaNumNisDocente());
}
```

ms | Javadoc | Declaration | Console | JUnit | Coverage

ter 0,04 seconds

/1 Errors: 0 Failures: 0

.com.inep.TEST.TesteNisDocente [Runner: JUnit 4] (0,000 s) Failure Trace

Figura 29 – Caso de teste para verificação do NIS

```
//REGISTRO 30 CAMPO 7 - VERIFICAÇÃO DO NIS DO DOCENTE
    public boolean verificaNumNisDocente() {
        Pattern pattern = Pattern.compile("[a-zA-Z]");
        Matcher matcher = pattern.matcher(this.id.getFkCodDocente());
        if(matcher.find() || this.id.getNumNisDocente().length() != 11
                || !validaPis(this.id.getNumNisDocente())){
            return false;
        }else
            return true;
    }

    public boolean validaPis(String pis){
        int i, soma = 0, sequencia = 9, resultado, resto;
        String caractere;
        for(i = 0; i < 10; i++){
            caractere = pis.substring(i, i+1);
            if(i == 0){ //SE PRIMEIRO CARACTERE
                soma = Integer.parseInt(caractere) * 3;
            }
            if(i == 1){ //SE SEGUNDO CARACTERE
                soma = (Integer.parseInt(caractere) * 2) + soma;
            }
            if(i != 0 && i != 1){ //SE QUALQUER OUTRO CARACTERE
                soma = (Integer.parseInt(caractere) * sequencia) + soma;
                sequencia = sequencia - 1;
            }
        }
        resto = soma % 11; //RESTO DA DIVISÃO DA VARIÁVEL SOMA POR 11
        resultado = 11 - resto;
        if(Integer.parseInt(pis.substring(10,11)) == resultado
                || (Integer.parseInt(pis.substring(10,11)) == 0 && (resultado == 10 || resultado == 11))){
            return true;
        }else{
            return false;
        }
    }
```

Figura 30 – Métodos para verificação do NIS

Para o Registro 30 foram implementados 14 casos de teste. A Tabela 17 informa os entregáveis da *Sprint* 4.

Tabela 17 – Entregáveis da Sprint 4

Classes na Camada de Modelo	Docente.java; NisDocente.java; GeradorNis.java; e TipoNecessidade.java.
Classes de Teste	TesteDocente.java; e TesteNisDocente.java.
Classes DAO – para verificação junto ao Banco de Dados	EntidadeDAO.java.
Número de Casos de Teste	14 casos de teste.
Tempo para Execução	2 dias.

6.5. *Sprint* 5 – Implementação dos Métodos do Registro 40

A Tabela 18 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 18 – *User Story* e tempo estimado para o Registro 40

User Story	*Story*: Criação dos métodos do Registro 40: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de docente: número do CPF, localização/zona de residência, CEP e endereço.
Tempo Estimado	15 dias.

Para implementação dos métodos do Registro 40, foi necessária uma classe java para descrição de objetos do tipo docente. Como a classe Docente.java já havia sido utilizada na implementação de alguns métodos do Registro 30, não foi necessária a estruturação da classe no projeto e a geração da respetiva tabela no banco de dados.

Com o uso do TDD, os métodos para verificação dos campos do Registro 40 foram criados, seguindo o mesmo processo que foi executado nos registros anteriores. A Tabela 19 traz os entregáveis dessa *Sprint*. Ao todo, foram criados 2 casos de teste.

Tabela 19 – Entregáveis da *Sprint* 5

Classes na Camada de Modelo	Docente.java.
Classes de Teste	TesteDocente.java.
Classes DAO – para verificação junto ao Banco de Dados	EstadoDAO.java; e MunicipioDAO.java.
Número de Casos de Teste	2 casos de teste.
Tempo para Execução	2 dias.

6.6. *Sprint* 6 – Implementação dos Métodos do Registro 50

A Tabela 20 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 20 – *User Story* e tempo estimado para o Registro 50

User Story	*Story*: Criação dos métodos do Registro 50: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de docente: escolaridade, formação superior (situação do curso superior, formação/complementação pedagógica, código do curso superior, ano de início e de conclusão do curso, tipo de instituição do curso superior, código da instituição do curso superior), pós-graduação, e outros cursos.
Tempo Estimado	15 dias.

Para implementação dos métodos do Registro 50, foram necessárias 8 classes java para descrição de objetos do tipo docente, escolaridade, classe de curso superior, área OCDE, instituição de ensino superior, formação superior do docente, pós-graduação e especialização.

Dessas classes, apenas a classe docente já havia sido estruturada no projeto. Assim, 7 classes foram estruturadas ao projeto e suas respectivas tabelas foram geradas no banco de dados.

O Registro 50 permite que sejam informadas até três formações superiores, ou seja, três possíveis cursos de graduação para cada docente. Para cada formação superior, devem ser informados 7 campos: situação do curso superior, formação/complementação pedagógica, código do curso superior, ano de início e ano de conclusão do curso, tipo de instituição do curso e código da instituição. Para este caso, foi necessária a implementação de apenas um método para cada campo, não sendo necessários novos métodos para verificação dos outros dois cursos de formação superior.

Para os campos de pós-graduação e de especialização, também foi necessário apenas um caso de teste e um método para verificação. A Figura 31 apresenta o caso de teste para este campo. A Figura 32 apresenta o método de verificação.

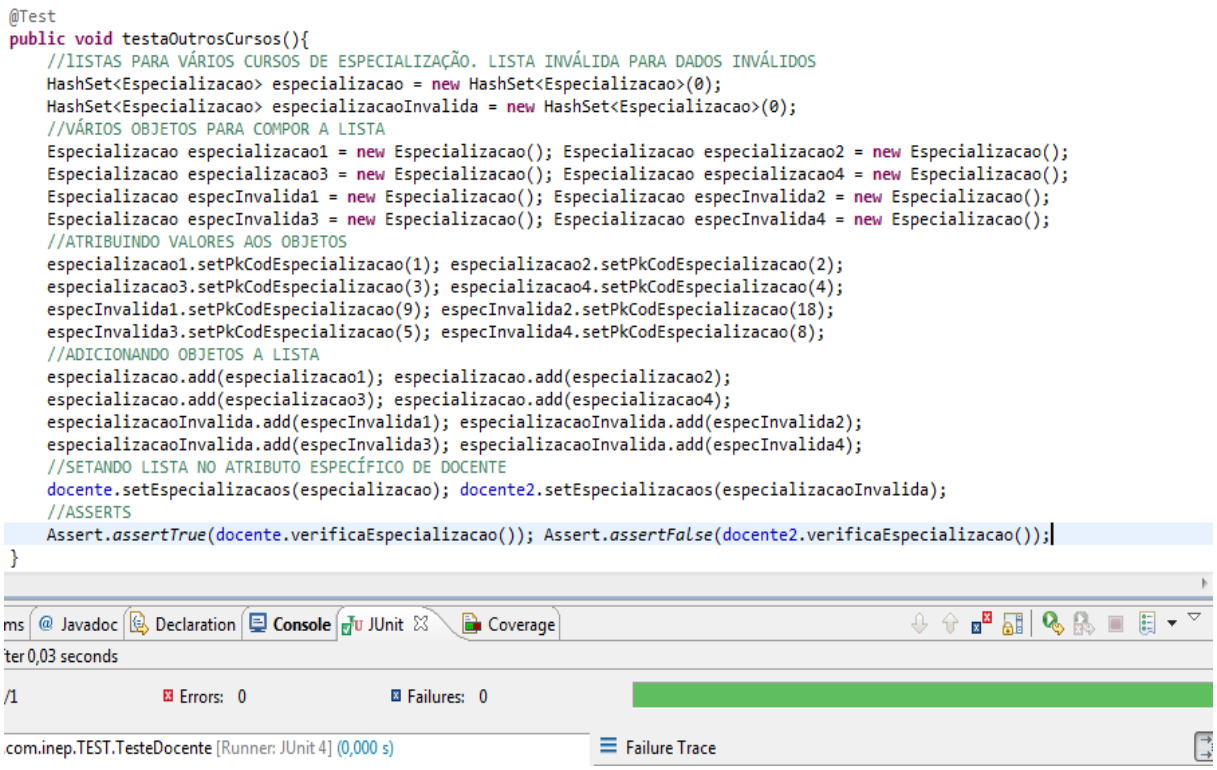

Figura 31 – Teste para verificação dos campos de especialização

```java
//REGISTRO 50 CAMPOS 31 A 46 - VERIFICAÇÃO DOS CURSOS DE ESPECIALIZAÇÃO
    public boolean verificaEspecializacao() {
        int erroCursos = 0;
        Iterator<Especializacao> i = this.especializacaos.iterator();
        while(i.hasNext()){
            Especializacao especializacao = (Especializacao) i.next();
            Pattern pattern = Pattern.compile(".*[0-9]");
            Matcher matcher = pattern.matcher(Integer.toString(especializacao.getPkCodEspecializacao()));
            if(!matcher.matches() || especializacao.getPkCodEspecializacao() == 9
              || especializacao.getPkCodEspecializacao() < 1 || especializacao.getPkCodEspecializacao() > 16
                  || !verificaCompatibilidadeCursos(this.especializacaos, especializacao.getPkCodEspecializacao())
                    || Integer.toString(especializacao.getPkCodEspecializacao()).length() > 2
                      || Integer.toString(especializacao.getPkCodEspecializacao()).length() < 1){
                erroCursos = erroCursos + 1;
            }
        }
        if(erroCursos > 0){
            return false;
        }else
            return true;
    }
//MÉTODO DE VERIFICAÇÃO DA COMPATIBILIDADE DOS CURSOS
    public boolean verificaCompatibilidadeCursos(
            Set<Especializacao> especializacaos2, int codigoCursoAtual) {
        int erroCompativel = 0;
        Iterator<Especializacao> i = this.especializacaos.iterator();
        while(i.hasNext()){
            Especializacao especializacao = (Especializacao) i.next();
            if(codigoCursoAtual == especializacao.getPkCodEspecializacao()){
                erroCompativel = erroCompativel + 1;
            }
        }
        if(erroCompativel > 1){ //CURSOS IGUAIS
            return false;
        }else return true;
    }
```

Figura 32 – Método para verificação da especialização

Para o Registro 50, foram necessários 3 casos de teste. Os entregáveis da *Sprint* 6 estão detalhados na Tabela 21

Tabela 21 – Entregáveis da *Sprint* 6

Classes na Camada de Modelo	Docente.java; Escolaridade.java; ClasseCurso.java; AreaOcde.java; Ies.java; DocenteFormSup.java; PosGraduacao.java; e Especializacao.java.
Classes de Teste	TesteDocente.java.
Classes DAO – para verificação junto ao Banco de Dados	AreaOcdeDAO.java; e IesDAO.java.
Número de Casos de Teste	3 casos de teste.
Tempo para Execução	1 dia.

6.7. *Sprint* 7 – Implementação dos Métodos do Registro 51

A Tabela 22 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 22 – *User Story* e tempo estimado para o Registro 51

User Story	*Story*: Criação dos métodos do Registro 51: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de docente (dados de docência): código da turma, função que exerce na escola/turma, situação funcional/regime de contratação/tipo de vínculo e código(s) da(s) disciplina(s) que leciona na turma.
Tempo Estimado	15 dias.

Para implementação dos métodos do Registro 51, foram necessárias 3 classes java para descrição de objetos do tipo docente, contratação do docente e disciplinas ministradas pelo docente na turma. As classes relacionadas à contratação do docente e às disciplinas ministradas pelo docente na turma foram estruturadas no projeto e suas classes no banco de dados foram geradas.

Para implementação dos métodos relacionados aos campos de disciplinas ministradas pelo docente na turma, foi necessária verificação das disciplinas ministradas na turma, verificação do tipo da turma e da sua etapa de ensino. Também foi necessário verificar se a disciplina informada estava na tabela auxiliar de disciplinas.

No total, foram implementados dois casos de teste. A Tabela 23 apresenta os entregáveis dessa *Sprint*.

Tabela 23 – Entregáveis da *Sprint* 7

Classes na Camada de Modelo	Docente.java; ContratacaoDocente.java; e DocenteDiscTurma.java.
Classes de Teste	TesteDocente.java; e

	TesteContratacaoDocente.java.
Classes DAO – para verificação junto ao Banco de Dados	TurmaDAO.java; e DiscTurmaDAO.java.
Número de Casos de Teste	2 casos de teste.
Tempo para Execução	8 dias.

6.8. *Sprint* 8 – Implementação dos Métodos do Registro 60

A Tabela 24 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 24 – *User Story* e tempo estimado para o Registro 60

User Story	*Story*: Criação dos métodos do Registro 60: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de aluno: nome completo, NIS, data de nascimento, sexo, cor/raça, filiação, nome da mãe, nome do pai, nacionalidade, país de origem, estado e município de nascimento, deficiência(s) (se possuir), recursos necessários para participação do aluno em avaliações do INEP (Prova Brasil, SAEB, outros).
Tempo Estimado	10 dias.

Para implementação dos métodos do Registro 60, foram necessárias 5 classes java para descrever objetos do tipo aluno, tipo de necessidade (deficiência), necessidade do aluno, recurso e recursos para necessidades. Todas essas classes foram estruturadas no projeto e suas respectivas tabelas foram geradas no banco de dados.

Os campos relacionados às necessidades do aluno foram testados por meio de um caso de teste. Para verificação dos dados informados nestes campos, foram implementados dois métodos que verificam as regras gerais e a compatibilidades entre necessidades (para alunos que possuem mais de uma deficiência). Para os campos de recursos, também foi necessário apenas um caso de teste. Neste caso, foi preciso verificar se a combinação entre necessidade e

recurso informada existia na tabela RECURSO_NECESSIDADE do banco de dados. A Figura 33 apresenta o teste utilizado para os campos de recursos. A Figura 34 apresenta os métodos de verificação desses campos nas classes AlunoNecessidade.java e RecursoNecessidadeDAO.java.

```
@Test
public void testaAlunoNecessidadeRecurso(){
    //TESTE RETORNO TRUE
    HashSet<RecursoNecessidade> recursosNecessidades = new HashSet<RecursoNecessidade>(0);

    RecursoNecessidade recursoNecessidade1 = new RecursoNecessidade();
    RecursoNecessidade recursoNecessidade2 = new RecursoNecessidade();
    recursoNecessidade1.setFkCodRecurso(1); recursoNecessidade1.setFkCodTipoNecessidade(1);
    recursoNecessidade2.setFkCodRecurso(2); recursoNecessidade2.setFkCodTipoNecessidade(2);
    recursosNecessidades.add(recursoNecessidade1); recursosNecessidades.add(recursoNecessidade2);
    alunoNecessidade.setRecursoNecessidades(recursosNecessidades);
    Assert.assertTrue(alunoNecessidade.verificaRecursoNecessidade());

    //TESTE RETORNO FALSO
    HashSet<RecursoNecessidade> recursosNecessidades2 = new HashSet<RecursoNecessidade>(0);

    RecursoNecessidade recursoNecessidade4 = new RecursoNecessidade();
    RecursoNecessidade recursoNecessidade5 = new RecursoNecessidade();
    recursoNecessidade4.setFkCodRecurso(3); recursoNecessidade4.setFkCodTipoNecessidade(3);
    recursoNecessidade5.setFkCodRecurso(2); recursoNecessidade5.setFkCodTipoNecessidade(2);
    recursosNecessidades2.add(recursoNecessidade4); recursosNecessidades2.add(recursoNecessidade5);
    alunoNecessidade2.setRecursoNecessidades(recursosNecessidades2);
    Assert.assertFalse(alunoNecessidade2.verificaRecursoNecessidade());
}
```

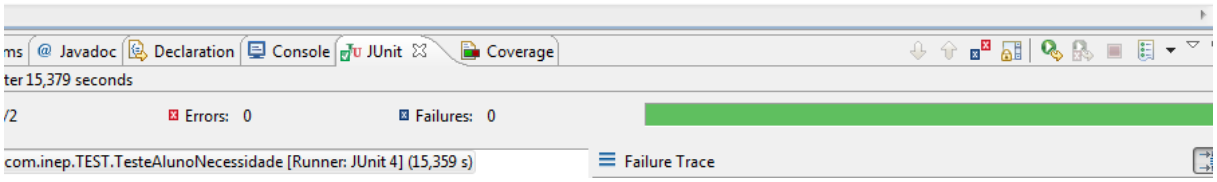

Figura 33 – Teste para verificação dos campos de recursos

```
//REGISTRO 60 CAMPOS 31 A 40 - VERIFICAÇÃO DO CÓDIGO DO ALUNO
    public boolean verificaRecursoNecessidade() {
        RecursoNecessidadeDAO recursoNecessidadeDAO = new RecursoNecessidadeDAO();
        int erro = 0;
        Iterator <RecursoNecessidade> i = this.recursoNecessidades.iterator();
        while(i.hasNext()){
            RecursoNecessidade recursoNec = (RecursoNecessidade) i.next();
            if(!recursoNecessidadeDAO.verificaRecursoNecessidade(recursoNec)){
                erro = erro + 1;
            }
        }
        if(erro > 0){
            return false;
        }else{
            return true;
        }
    }
}

//VERIFICA EXISTÊNCIA DO RECURSO/NECESSIDADE
    public boolean verificaRecursoNecessidade(RecursoNecessidade recursoNecessidade) {
        boolean retorno = false;
        try {
            EntityManagerFactory factory = Persistence.createEntityManagerFactory("entidade");
            EntityManager manager = factory.createEntityManager();
            RecursoNecessidade encontrado = manager.find(RecursoNecessidade.class, recursoNecessidade);
            manager.clear();
            factory.close();
            if(encontrado != null){
                retorno = true;
            }
        } catch (Exception e) {
            System.out.println("RECURSO NÃO COMPATÍVEL COM NECESSIDADE.");
            System.out.println(e);
            retorno = false;
        }
        return retorno;
    }
```

Figura 34 – Métodos para verificação dos campos de recursos

Ao todo, para o Registro 60, foram implementados 19 casos de teste. A Tabela 25 apresenta os entregáveis dessa *Sprint*.

Tabela 25 – Entregáveis da *Sprint* 8

Classes na Camada de Modelo	Aluno.java; NisAluno.java; AlunoNecessidade.java; TipoNecessidade.java; Recurso.java; e RecursoNecessidade.java.
Classes de Teste	TesteAluno.java; TesteNisAluno.java; e TesteAlunoNecessidade..java.
Classes DAO – para verificação junto ao Banco de Dados	RecursoNecessidadeDAO.java.
Número de Casos de Teste	19 casos de teste.
Tempo para Execução	6 dias.

6.9. *Sprint* 9 – Implementação dos Métodos do Registro 70

A Tabela 26 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 26 – *User Story* e tempo estimado para o Registro 70

User Story	*Story*: Criação dos métodos do Registro 70: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de aluno: número de identidade, complemento da identidade, órgão emissor, estado de emissão da identidade, data de expedição, dados de certidão civil, número do CPF, documento estrangeiro/passaporte, justificativa para falta de documentação do aluno (quando não houver dados de documentos preenchidos), localização/zona de residência, CEP e endereço.

Tempo Estimado	10 dias.

Para implementação dos métodos do Registro 70, foram necessárias 3 classes java para descrever objetos do tipo aluno, órgão emissor e cartório. As classes OrgaoEmissor.java e Cartorio.java foram estruturadas no projeto e suas respectivas tabelas foram geradas no banco de dados.

Da mesma forma que foi feito nas *Sprints* anteriores, foi utilizado o TDD para implementação dos métodos de verificação dos campos. Os campos órgão emissor da identidade, uf da identidade e código do cartório fizeram uso de tabelas auxiliares para verificação dos valores possíveis para esses campos. Os campos que detalham a localização residencial dos alunos foram feitos da mesma maneira que os campos de residência dos docentes.

Foram implementados ao todo 22 casos de testes para criação dos métodos de verificação do Registro 70. A Tabela 27 mostra os entregáveis dessa *Sprint*.

Tabela 27 – Entregáveis da *Sprint* 9

Classes na Camada de Modelo	Aluno.java; Cartorio.java; e OrgaoEmissor.java.
Classes de Teste	TesteAluno.java;
Classes DAO – para verificação junto ao Banco de Dados	CartorioDAO.java; e OrgaoEmissorDAO.java.
Número de Casos de Teste	22 casos de teste.
Tempo para Execução	5 dias.

6.10. *Sprint* 10 – Implementação dos Métodos do Registro 80

A Tabela 28 traz a *user story* para esta *Sprint*, bem como o tempo estimado para sua implementação.

Tabela 28 – *User Story* e tempo estimado para o Registro 80

User Story	*Story*: Criação dos métodos do Registro 80: Eu, como desenvolvedor, pretendo criar os métodos que verifiquem os seguintes dados de aluno (matrícula): código da turma, turma unificada, etapa do aluno em turma multietapa/multi/correção de fluxo/EJA presencial e semipresencial – anos iniciais e finais/educação profissional mista – concomitante e subsequente, escolarização em outro espaço (para alunos que recebem escolarização em espaço diferente da escola), transporte escolar público, poder público responsável pelo transporte escolar e tipo de veículo.
Tempo Estimado	10 dias.

Para implementação dos métodos do Registro 80, foram necessárias 3 classes java para descrever objetos do tipo matrícula, forma de ingresso do aluno de escola federal e tipo de transporte escolar público. Essas classes foram estruturadas no projeto e suas respectivas tabelas foram geradas no banco de dados.

Ao inserir o código da turma na qual se pretende vincular o aluno, foi necessário uma verificação da etapa de ensino informada para a turma. Caso a turma fosse unificada (creche e pré-escola na mesma turma) ou multietapa ou correção de fluxo ou EJA presencial e semipresenciais anos iniciais e anos finais ou educação profissional mista (concomitante e subsequente), é necessário informar a etapa do aluno nesta turma.

Ao implementar o método para verificação do tipo de transporte escolar público, utilizado pelo aluno para chegar à escola, foi necessário verificar o valor informado no campo Transporte escolar público (Registro 80 Campo 11) que informa se o aluno precisa ou não de transporte público. As Figuras 35 e 36 apresentam o teste e o método implementados para executar esta verificação.

```
@Test
public void testaTipoTransportePublico(){
    matriculaTrue.setIdNTEP(1); //ALUNO PRECISA DE TRANSPORTE ESCOLAR
    HashSet<TipoTransporte> tipoTransportes = new HashSet<TipoTransporte>(0);
    //OBJETOS PARA OS TIPOS DE TRANSPORTES UTILIZADO PELO ALUNO.
    TipoTransporte tipoTransporte1 = new TipoTransporte();
    TipoTransporte tipoTransporte2 = new TipoTransporte();
    TipoTransporte tipoTransporte3 = new TipoTransporte();
    tipoTransporte1.setPkCodTipoTransporte(1);
    tipoTransporte2.setPkCodTipoTransporte(2);
    tipoTransporte3.setPkCodTipoTransporte(4);
    Pattern pattern = Pattern.compile(".*[0-9]");
    Matcher matcher = pattern.matcher(Integer.toString(tipoTransporte3.getPkCodTipoTransporte()));
    tipoTransportes.add(tipoTransporte1); tipoTransportes.add(tipoTransporte2); tipoTransportes.add(tipoTransporte3);
    matriculaTrue.setTipoTransportes(tipoTransportes);
    Assert.assertTrue(matcher.matches());
    //CHAMADA DO PRINCIPAL MÉTODO DE VERIFICAÇÃO
    Assert.assertTrue(matriculaTrue.verificaTipoTransporte());
}
```

ms | Javadoc | Declaration | Console | JUnit | Coverage

ter 18,831 seconds

0/10 Errors: 0 Failures: 0

com.inep.TEST.TesteMatricula [Runner: JUnit 4] (18,789 s) Failure Trace

Figura 35 – Teste para verificação do tipo de transporte escolar público

```
//REGISTRO 80 CAMPOS 13 A 23 - VERIFICAÇÃO DO TIPO DE TRANSPORTE ESCOLAR PÚBLICO
    public boolean verificaTipoTransporte() {
        boolean erro = false;
        int itemCorreto = 0;
        Iterator<TipoTransporte> i = this.tipoTransportes.iterator();
        while(i.hasNext()){
            TipoTransporte tipoTransporte = (TipoTransporte) i.next();
            int tipo = tipoTransporte.getPkCodTipoTransporte();
            Pattern pattern = Pattern.compile(".*[0-9]");
            Matcher matcher = pattern.matcher(Integer.toString(tipo));
            if(tipo > 0 && tipo < 12  && matcher.matches()
                    && Integer.toString(tipo).length() <= 2){
                itemCorreto = itemCorreto + 1;
            }else{
                erro = true;
            }
        }
        if(itemCorreto > 0 && erro == false
                && this.tipoTransportes.size() < 4 && this.idNTEP == 1){
            return true;
        }else{
            return false;
        }
    }
```

Figura 36 - Método para verificação do tipo de transporte escolar público

Foram implementados ao todo 10 casos de testes para criação dos métodos de verificação do Registro 70. A Tabela 29 mostra os entregáveis dessa *Sprint*.

Tabela 29 – Entregáveis da *Sprint* 10

Classes na Camada de Modelo	Matricula.java; FormaIngresso.java; e TipoTransporte.java.
Classes de Teste	TesteMatricula.java;
Classes DAO – para verificação junto ao Banco de Dados	TurmaDAO.java; ModEtapaEnsDAO.java; e TipoTransporteDAO.java.

Número de Casos de Teste	10 casos de teste.
Tempo para Execução	5 dias.

6.11. Testes sobre a Camada de Modelo

Conforme o processo definido para implementação da camada de modelo, usou-se o TDD para criação dos métodos de verificação dos campos do LIE. Para medir a cobertura de código, utilizou-se a ferramenta EclEmma. Com base na última medição realizada, 80,5% do código está coberto por testes. A Figura 37 mostra este dado. A parte do código que não ficou coberta por testes é composta pelos métodos *getters* e *setters* dos atributos das classes java.

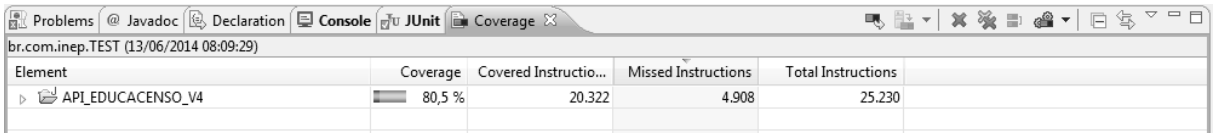

Figura 37 – Cobertura do código

Além dos testes realizados por meio do TDD, foram feitos testes sobre arquivos de importação gerados pela classe GeraArquivo.java, criada na camada de modelo. Com base nas regras gerais do LIE, escolas paralisadas e extintas podem preencher apenas alguns dados do registro 00 e do registro 10, ou seja, no arquivo de importação, apenas esses registros devem existir. Já escolas em funcionamento, todos os registros constantes no LIE devem ser preenchidos.

Com base nisso, foi gerado um arquivo de importação de uma escola paralisada, cadastrada no banco de dados após passar por todos os métodos de verificação de campos implementados na camada de modelo e outro arquivo de importação de uma escola em funcionamento, também cadastrada no banco de dados após passar pelos métodos. A Seção 6.11.1 mostra os resultados dos testes com a escola paralisada e a Seção 6.11.2 mostra os resultados dos testes com a escola em funcionamento.

6.11.1. Teste do Arquivo de Importação de uma Escola Paralisada

Foram instanciados os objetos relativos ao Registro 00 e ao Registro 10 nas classes Registro00Main e Registro10Main, respectivamente. Dentre as características dessa escola, destacam-se: escola privada e paralisada, localizada no estado do Paraná em zona urbana.

Após instanciar os objetos e definir valores para os campos, os dados foram verificados por meio dos métodos implementados. Após a verificação, os objetos foram persistidos no banco de dados.

Através da classe GeraArquivoTxt.java, foi gerado um arquivo no formato especificado pelo LIE para a escola citada. Após isso, o arquivo foi submetido às verificações do Migradados. Após as verificações, foi notificado que o arquivo havia sido validado com sucesso e que poderia ser enviado para o INEP, ou seja, os campos estavam corretamente informados, de acordo com as regras definidas no LIE. A Figura 38 mostra o resultado obtido.

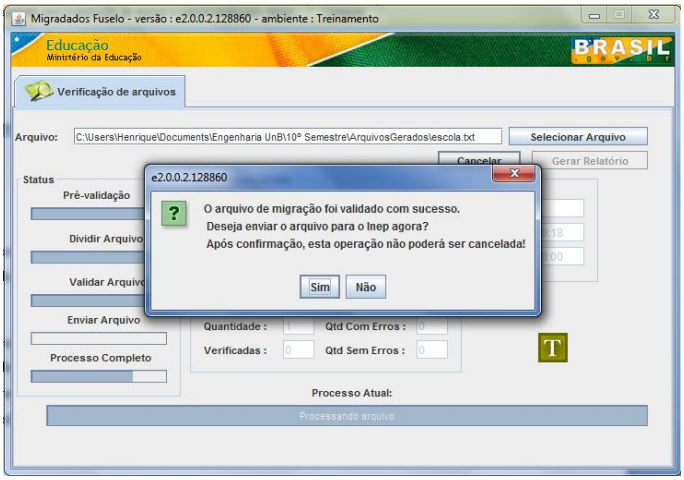

Figura 38 – Resultado da verificação do arquivo no migradados

6.11.2. Teste do Arquivo de Importação de uma Escola em Funcionamento

Foram instanciados os objetos necessários para todos os registros do LIE. Os dados foram verificados, utilizando os métodos implementados. Durante este processo de verificação, foram corrigidos alguns erros pontuais no código, não encontrados durante a implementação. Com o objetivo de testar o arquivo de uma escola em funcionamento, foram instanciados objetos com atributos de uma escola com essa característica. Ao fim, vários objetos foram persistidos no banco de dados.

Dentre as características da escola, tem-se: escola em funcionamento, de dependência administrativa municipal, funciona em prédio escolar e não compartilha o prédio. Oferece atividade complementar e modalidade regular de ensino. Oferece as etapas do ensino fundamental de 9 anos. Possui uma turma de atividade complementar, uma turma de 1º ano, outra de 2º ano e mais duas de 9º ano. Possui 6 alunos, vinculados à essas turmas.

Através da classe GeraArquivoTxt.java, foi gerado um arquivo no formato especificado pelo LIE, com os dados da escola. Após este processo, o arquivo foi submetido no migradados para verificação dos dados da escola, com vistas às regras apresentadas pelo LIE. O resultado obtido foi positivo, igual ao apresentado após a submissão do arquivo da escola paralisada.

Com estes resultados, vê-se que os métodos implementados estão realizando as verificações conforme especificado pelo LIE. De qualquer forma, os arquivos das escolas gerados possuem dados de uma escola pequena, comparada a escolas da realidade brasileira. Antes de usar a biblioteca efetivamente para construção de sistemas escolares, seria necessária a criação de escolas com maior número de dados e variações, fazendo assim uma homologação mais completa da camada de modelo, como é feito normalmente nos softwares atuais.

6.12. Gerenciamento do Projeto

Para que fosse possível acompanhar a situação do projeto durante sua execução, foram criadas as seguintes métricas:

- Número de campos – Número total de campos descritos no LIE, ou seja, a soma de todos os campos dos registros;
- Número de tabelas envolvidas – Número total de tabelas envolvidas para implementação das regras expostas no LIE;
- Número de classes envolvidas – Número total de classes envolvidas para implementação das regras expostas no LIE;
- Número de regras – Número de regras expostas no LIE;
- Porcentagem de campos com regras gerais implementadas – Compreende o número de campos com regras gerais implementadas dividido pelo número de campos total;
- Porcentagem de campos com regras específicas implementadas – Compreende o número de campos com regras específicas implementadas dividido pelo número de campos total;

- Porcentagem de registros com regras gerais implementadas – Compreende o número de registros com regras gerais implementadas dividido pelo número total de registros;
- Porcentagem de registros com regras específicas implementadas – Compreende o número de registros com regras específicas implementadas dividido pelo número total de registros;
- Porcentagem de regras gerais implementadas – Compreende o número de regras gerais implementadas dividido pelo número de regras total; e
- Porcentagem de regras específicas implementadas – Compreende o número de regras específicas implementadas dividido pelo número de regras total.

Durante a execução do projeto, foram feitas várias medições para atualizar as medidas para essas métricas. Para acompanhar o estado do projeto em termos do número de campos implementados, foi utilizado um gráfico de linha conforme o apresentado pela Figura 39.

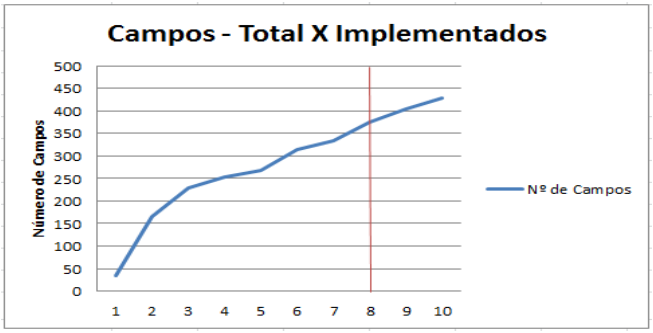

Figura 39 – Gráfico de Linha para Acompanhamento do Estado do Projeto

A Fase B deste projeto foi divida nas 12 seguintes atividades, sendo que a atividade 12ª está além dos resultados deste trabalho:

1. Estruturar as classes, implementar regras e refatorar métodos do registro 00;
2. Estruturar as classes, implementar regras e refatorar métodos do registro 10;
3. Estruturar as classes, implementar regras e refatorar métodos do registro 20;
4. Estruturar as classes, implementar regras e refatorar métodos do registro 30;
5. Estruturar as classes, implementar regras e refatorar métodos do registro 40;
6. Estruturar as classes, implementar regras e refatorar métodos do registro 50;

7. Estruturar as classes, implementar regras e refatorar métodos do registro 51;
8. Estruturar as classes, implementar regras e refatorar métodos do registro 60;
9. Estruturar as classes, implementar regras e refatorar métodos do registro 70;
10. Estruturar as classes, implementar regras e refatorar métodos do registro 80;
11. Escrita do TCC; e
12. Escrita e submissão de artigo para workshop/conferência de informática na educação (ou similar).

A Tabela 30 informa o cronograma base planejado para este trabalho.

Tabela 30 – Cronograma do Trabalho

Atividades	**Dezembro/2013**	**Março/2014**	**Abril/2014**	**Maio/2014**	**Junho/2014**	**Julho/2014**
0	X					
1	X					
2	X					
3		X				
4		X				
5			X			
6			X			
7				X		
8				X		
9				X		

10					X	
11						X

Para acompanhamento da execução das atividades da Fase B, foi utilizado um gráfico de linhas, que apresenta a execução planejada para as atividades no tempo e a execução realizada durante o projeto. A Figura 40 apresenta este gráfico.

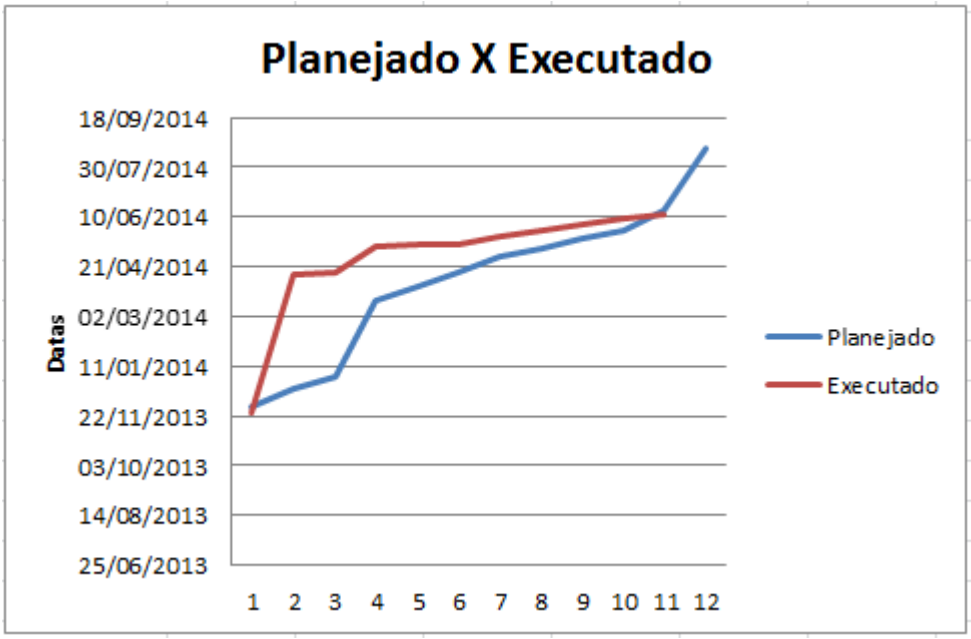

Figura 40 – Execução das Atividades da Fase B X Planejamento Inicial

Com base no gráfico apresentado pela Figura 32, a execução das atividades foi, de modo geral, atrasada. O atraso se deve, principalmente pela falta de conhecimento da linguagem java obtido com a execução do projeto. Nota-se que após uma fase inicial de implementação atrasada devido ao aprendizado, a partir da atividade 5, alcançou-se uma maior produtividade na implementação do projeto. Ao final, concluíram-se as atividades no tempo previsto e os objetivos relacionados à implementação da camada de modelo foram alcançados.

7. Considerações Finais

Uma boa gestão dos dados escolares pode ajudar tanto processos internos de instituições de ensino, como também processos externos, considerando o Censo Escolar e a formulação de políticas públicas. Os dados que são coletados pelo Censo da Educação Básica são definidos pelo Ministério da Educação e por requisição de órgãos públicos que necessitam de determinadas informações escolares.

As regras de validação de cada dado coletado pelo Censo, bem como aquelas que regem o negócio estão implementadas no sistema Educacenso. Dessa forma, todas as instituições devem obedecer às regras do Censo, no momento de respondê-lo. O Layout de Importação/Exportação reúne grande parte dessas regras e é usado pelas instituições de ensino na implementação de seus sistemas.

Apesar da disponibilização do Layout de Importação/Exportação, muitos sistemas escolares não são construídos com base nas regras do sistema Educacenso. Este fato faz com que o fornecimento de dados ao Censo Escolar seja dificultado, devido ao grande número de erros que ocorrem ao se enviar os arquivos de migração para o Educacenso.

Tendo em vista esses erros, além de outros que acontecem no processo, este TCC teve como objetivo a implementação de uma camada de modelo de um software que obedeça as regras dispostas no Layout de Importação/Exportação, de forma que os sistemas escolares que utilizarem essa camada fiquem mais semelhantes ao sistema Educacenso.

Para que esta camada de modelo fosse implementada, foram utilizados o padrão MVC, a metodologia de desenvolvimento SCRUM, o desenvolvimento orientado a testes, além das ferramentas TOAD, SQL Developer, Eclipse e EclEmma. Também foi utilizada a especificação JPA, visando o uso da modelagem objeto relacional.

Primeiramente foi feita uma engenharia reversa do Sistema Educacenso e do seu banco de dados com o objetivo de entender seu funcionamento, fluxos principais e alternativos e tabelas onde os dados são armazenados. Após isso, foi criado um banco de dados, com a mesma estrutura do banco do Sistema Educacenso, para ser entrada para geração de classes java, realizando o mapeamento objeto relacional, necessário para mapear um projeto orientado a objetos com o banco de dados relacional criado. Após a geração das

classes, foi criado um novo projeto onde, conforme as necessidades dos registros do LIE, eram estruturadas classes java e implementados os métodos.

A implementação foi feita seguindo a ordem dos registros expostos no Layout de Importação/Exportação. Ao final da implementação das regras de um registro, as atividades de estruturação de classes, implementação de métodos e refatoração de código foram executadas para o registro seguinte.

Para garantir uma boa cobertura de código, foi utilizado o EclEmma durante a implementação, para acompanhar a porcentagem do código que estava coberta por testes. Além da cobertura, foram realizados vários testes sobre a camada durante a sua implementação. Por meio do uso do TDD, testou-se entradas válidas e inválidas com o objetivo de garantir a correta execução e função dos métodos.

Além do TDD, após a implementação das camadas, foram criadas duas escolas, uma paralisada e outra em funcionamento. Antes de persistir os objetos dessas escolas no banco, os dados foram verificados utilizando os métodos implementados. Após a verificação e a persistência, foram gerados arquivos de importação, segundo as especificações do LIE. Estes arquivos foram submetidos no migradados para verificação dos dados das escolas. Ao final, obtiveram-se resultados positivos com relação aos dados, garantindo o correto funcionamento dos métodos.

Apesar desses testes, acredita-se que a camada pode ser testada com um maior número de dados e variações. Anualmente o Educacenso passa por uma fase de homologação que dura pelo menos dois meses. Acredita-se que com um maior número de testes, a camada de modelo implementada neste TCC terá maior garantia frente ao número possível de variações de dados que são encontradas no Censo Escolar.

Como trabalhos futuros, pretende-se implementar o Layout do Arquivo de Importação/Exportação para a fase de Situação do Aluno do Censo Escolar, quando são informados o rendimento e os movimentos dos alunos durante o ano letivo. Também se pretende elaborar e submeter um artigo sobre este trabalho de conclusão de curso em workshop/conferência de informática na educação (ou similar).

Referências Bibliográficas

ANICHE, Maurício, 2012. Como a prática de TDD influencia o projeto de classes em sistemas orientados objetos. Universidade de São Paulo. São Paulo. Disponível em: < http://www.aniche.com.br/wp-content/uploads/2013/04/master-dissertation.pdf > Acessado em: 17 nov. 2013.

ARONSON, R. B. 1996 apud FERNEDA, Amauri, 1999. Integração Metrologia, CAD e CAM: Uma contribuição ao estudo de engenharia reversa. Universidade de São Paulo, São Paulo. Disponível em: < www.teses.usp.br/teses/disponiveis/18/18135/tde.../TDE_Amauri.pdf > Acessado em: 20 jul. 2013.

BECK, 2004 apud ELIAS, Guilherme; WILDT, Daniel, 2008. Métricas para Melhoria Contínua de Còdigo – Um Estudo de Caso com Java. Faculdade Cenescista Nossa Senhora dos Anjos. Rio Grande do Sul. Disponível em: < file:///C:/Users/henrique.santos/Downloads/SEMINFO2008_GuilhermeElias_DanielWildt_MetricasMelhoriaContinuaCodigo.pdf> Acessado em: 03 jun. 2014.

BENEGA, Thiago, 2010. Padrões de Projeto em Modelagem Orientada a Objetos Persistida em Bano de Dados Relacional. Ceará. Disponível em: <http://www.ffb.edu.br/sites/default/files/tcc-20101-thiago-vicente-benega.pdf> Acessado em: 22 jul. 2013.

BISSI, Wilson, 2007. Scrum – Metodologia de Desenvolvimento Ágil. Centro Universitário de Maringá. Paraná. Disponível em: <http://www.proa.pa.gov.br/bitstream/prodepa/34/1/scrum.pdf> Acessado em: 22 nov. 2013.

BRASIL. Decreto nº 19.402/1930. 1930. Disponível em: <http://portal.mec.gov.br/arquivos/pdf/d19402.pdf> Acessado em: 16 jul. 2013.

BRASIL. Decreto nº 6.320/1930. 2007. Disponível em: <http://intramec.mec.gov.br/index.php?option=com_docman&gid=349&task=doc_view> Acessado em: 16 jul. 2013.

BRASIL. Decreto-Lei nº 580/1938. 1938. Disponível em: <http://www.histedbr.fae.unicamp.br/navegando/fontes_escritas/5_Gov_Vargas/decreto-lei%200580-1938%20inep.htm> Acessado em: 16 jul. 2013.

BRASIL. Lei nº 9448/1997. 1997 Disponível em: <http://www.planalto.gov.br/ccivil_03/Leis/L9448.htm> Acessado em: 16 jul. 2013.

BRASIL. Decreto nº 6.425/2008. 2008. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2008/Decreto/D6425.htm> Acessado em: 17 jul. 2013.

BRASIL. Decreto nº 6.317/2007. 2007. Disponível em: <http://www.planalto.gov.br/ccivil_03/_Ato2007-2010/2007/Decreto/D6317.htm> Acessado em: 21 nov. 2013.

BRASIL. INEP. Resumo Técnico. Censo Escolar da Educação Básica 2012. Disponível em: <http://download.inep.gov.br/educacao_basica/censo_escolar/resumos_tecnicos/resumo_tecnico_censo_educacao_basica_2012.pdf> Acessado em: 16 jul. 2013.

BRASIL. INEP. Proposta de Reestruturação – DEED. 2011. Disponível em: <http://inepnet.inep.gov.br/index.php?option=com_phocadownload&view=sections&Itemid=34> Acessado em: 16 jul. 2013.

BRASIL. INEP. Censo Escolar da Educação Básica 2013. 2013. Disponível em: <http://download.inep.gov.br/educacao_basica/Educacenso/documentos/2013/folder_censo_escolar_educacao_basica_2013.pdf> Acessado em: 17 jul. 2013.

BRASIL. INEP. Layout de Importação/Exportação Educacenso 2013. Disponível em: < http://sitio.educacenso.inep.gov.br/migracao > Acessado em: 21 nov. 2013.

CAETANO, Daniel, 2013. Padrão MVC e DAO. Disponível em: <www.caetano.eng.br/aulas/2013b/getfile.php?fn=psw_ap06.pdf> Acessado em: 21 nov. 2013.

COPELAND, 2004 apud RINCON, 2011. Qualidade de Conjuntos de Teste de Software de Código Aberto: Uma Análise Baseada em Critérios Estruturais. Universidade Federal de Goiás. Goiás. Disponível em: < http://www.inf.ufg.br/mestrado/sites/www.inf.ufg.br.mestrado/files/uploads/Dissertacoes/AndreMesquita.pdf> Acessado em: 03 jun. 2014.

DEITEL, H. M, 2006. C++ como programar. 5. Ed. São Paulo. Pearson Education do Brasil.

ECLIPSE.ORG, 2006. *Eclipse Platform Technical Overview*. Disponível em: < http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf > Acessado em: 17 nov. 2013.

ECLEMMA.ORG, s/d. *Java Code Coverage for Eclipse*. Disponível em: < http://www.eclemma.org/> Acessado em: 05 jun. 2014.

FONSECA, Marcos; ALVES, André, s/d. MVC e Framework. Goiás. Disponível em: < http://www.cpgls.ucg.br/ArquivosUpload/1/File/V%20MOSTRA%20DE%20PRODUO%20CIENTIFICA/EXATAS/2-.PDF > Acessado em: 16 nov. 2013.

GASPARETO, Otávio, s/d. Test Driven Development. Universidade Federal do Rio Grande do Sul. Rio Grande do Sul. Disponível em: < http://www.inf.ufrgs.br/~cesantin/TDD-Otavio.pdf > Acessado em: 17 nov. 2013.

GUERRA, Rafael, 2008. Mapamento objeto relacional: Um estudo de caso utilizando o Hibernate. São Paulo. Disponível em: <fatecindaiatuba.edu.br/reverte/index.php/revista/article/download/9/10> Acessado em: 22 jul. 2013.

JUNIOR, Antonio; SOUZA Diego; et al., 2005. Engenharia Reversa. Universidade Federal Fluminense, Rio de Janeiro. Disponível em: < http://www2.ic.uff.br/~otton/graduacao/informaticaI/apresentacoes/eng_reversa.pdf> Acessado em: 20 jul. 2013.

JUNIOR, Roberto, 2011. Oracle SQL Developer: instalação, configuração, personalização e atualização. Disponível em: < http://imasters.com.br/artigo/19522/ > Acessado em: 17 nov. 2013.

PINHEIRO, Adna; at al. 2006. Sistema de Coleta on-line do Censo Escolar da Educação Básica – Educacenso. Concurso Inovação na Gestão Pública Federal. Disponível em: < http://inovacao.enap.gov.br/index.php?option=com_docman&task=doc_view&gid=313> Acessado em: 21 nov. 2013.

REENSKAUG, Trygve, 1979. Models – Views – Controllers. University of Oslo, Noroega. Disponível em: <http://folk.uio.no/trygver/1979/mvc-2/1979-12-MVC.pdf> Acessado em: 16 nov. 2013.

RIBEIRO, Rafael, s/d. Garantia de Processo. Leis de Lehman. Manutenção de Sofware. Disponível em: <http://www.rafaeldiasribeiro.com.br/downloads/testesw5.pdf>. Acessado em: 21 nov. 2013.

SIMÕES, Rhodney; SANTOS, Marcelo, 2009. Utilização do padrão MVC em aplicações ricas para internet com modelagem em webml. Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica, Pará. Disponível em: < http://connepi2009.ifpa.edu.br/connepi-anais/artigos/102_4209_2048.pdf > Acessado em: 16 nov. 2013.

ROSS, D.T., 1985. Applications and Extensions of SADT. Computer, vol. 18. Disponível em: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1662862&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F2%2F34804%2F01662862 > Acessado em: 21 nov. 2013.

SCHWABER, Jeff; SUTHERLAND, Jeff, 2011. Guia do Scrum. Disponível em: < https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20Portuguese%20BR.pdf> Acessado em: 17 nov. 2013.

SOMMERVILLE, Ian, 2003. Engenharia de Software. 6. Ed. São Paulo. Pearson Addison Wesley.

TORRES, Alexandre, 2009. MD-JPA: Um perfil UML para modelagem do mapeamento objeto-relacional com JPA em um abordagem dirigida por modelos. Universidade Federal do Rio Grande do Sul. Rio Grade do Sul. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/17797/000724396.pdf?sequence=1> Acessado em: 16 nov. 2013.

Anexo 1

O Anexo 1 traz a descrição das Tabelas do Banco de Dados do Sistema Educacenso.

Tabela	Descrição
TAB_ADMITIDO_APOS	Possui 14 colunas e tem a função de informar quais foram os alunos que entraram na escola depois da data de referência do censo escolar. Seus principais dados armazenados são: código do aluno, código da entidade, código da matrícula, códigos da modalidade e etapa de ensino e outros.
TAB_ALUNO	Possui 34 colunas e tem a função de informar dados gerais dos alunos. Seus principais dados são: código do aluno, nome, endereço, filiação, nacionalidade, deficiência e outros.
TAB_ALUNO_NECESSIDADE	Possui 2 colunas e tem a função de informar o código da necessidade (deficiência) do aluno
TAB_ANO_CENSO	Possui 4 colunas e tem a função de armazenar informações do ano do censo, como status, data de início e data de término.
TAB_ALUNO_NEC_RECURSO_NEC	Possui 3 colunas e tem a função de informar o código da necessidade (deficiência) do aluno juntamente com o recurso necessário para realização de provas aplicadas pelo INEP como Prova Brasil.
TAB_AREA_CURSO_ED_PROF	Possui 2 colunas e tem a função de informar o código e a descrição da área do curso de educação profissional.
TAB_AREA_OCDE	Possui 4 colunas e tem a função de informar o código, a classe e o nome da área do curso de

	educação superior. Relaciona-se com a tabela TAB_DOCENTE.
TAB_AREA_TIPO_ATIVIDADE	Possui 2 colunas e tem a função de informar o código e a descrição da área do tipo da atividade.
TAB_CLASSE_CURSO	Possui 2 colunas e tem a função de informar o código e a descrição da classe de um curso.
TAB_CONTRATACAO_DOCENTE	Possui 3 colunas e tem a função de informar o código da entidade, do docente vinculado a entidade e da forma do tipo de contratação do docente.
TAB_CURSOS_ED_PROF	Possui 3 colunas e tem a função de informar o código do curso e da área do curso de educação profissional, juntamente com sua descrição.
TAB_DADO_DOCENCIA	Possui 8 colunas e tem a função de informar os dados de docência, ou seja, os vínculos entre entidade, turma e docente.
TAB_DADO_ESCOLA	Possui 80 colunas e tem a função de informar outros dados de escola, como dados de escola mantenedora, número de salas existentes, categoria da escola privada, convenio com o poder público, abastecimento de água, abastecimento de energia elétrica, esgoto sanitário, destinação do lixo e outros.
TAB_DIAS_SEMANA	Possui 2 colunas e tem a função de informar o código e o nome dos dias da semana.
TAB_DIRIGENTE	Possui 9 colunas e tem a função de informar dados do gestor escolar.

TAB_DISCIPLINA	Possui 3 colunas e tem a função de informar o código e o nome das disciplinas.
TAB_DISCIPLINA_ETAPA_ENS	Possui 3 colunas e tem a função de informar as disciplinas possíveis para cada modalidade/etapa de ensino.
TAB_DISCIPLINA_TURMA	Possui 7 colunas e tem a função de informar quais as disciplinas que são ministradas em determinada turma.
TAB_DISTRITO	Possui 4 colunas e tem a função de informar dados de distritos brasileiros.
TAB_DOCENTE	Possui 27 colunas com dados que dizem respeito ao código, nome, endereço, filiação, nacionalidade, deficiência e outros do docente.
TAB_DOCENTE_DISC_TURMA	Possui 6 colunas com dados que dizem respeito ao as disciplinas ministradas por determinados docentes em determinadas turmas.
TAB_DOCENTE_ENTIDADE	Possui 3 colunas e tem a função de informar os docentes que foram informados via migração de dados.
TAB_DOCENTE_ESPECIALIZACAO	Possui 2 colunas com dados que dizem respeito aos cursos de especialização dos docentes.
TAB_DOCENTE_FORM_SUP	Possui 7 colunas e tem a função de informar dados da formação superior dos docentes.
TAB_DOCENTE_NECESSIDADE	Tem 2 colunas e tem a função de informar as necessidades (deficiências) de docentes enquadrados neste grupo.

TAB_DOCENTE_POS_GRADUACAO	Tem 2 colunas e tem a função de informar dados de pós-graduação dos docentes.
TAB_ENTIDADE	Possui 35 colunas com dados que dizem respeito ao código, nome, endereço, dependência administrativa e contato da escola e outros.
TAB_ENTIDADE_SUPERIOR	Tem 2 colunas e tem a função de informar o código da escola e sua respectiva entidade superior.
TAB_ESCOLA_COMPARTILHADA	Tem 2 colunas e informa o código da escola, juntamente com o código daquela com a qual compartilha prédio escolar.
TAB_ESCOLA_DEPENDENCIA	Tem 3 colunas e tem a função de informar as dependências existentes na escola.
TAB_ESCOLA_EQUIPAMENTO	Tem 4 colunas e tem a função de informar os equipamentos existentes na escola.
TAB_ESCOLA_FUNC	Tem 3 colunas e tem a função de informar o local de funcionamento da escola.
TAB_ESCOLA_MANTENEDORA	Tem 3 colunas e tem a função de informar dados da mantenedora da escola privada.
TAB_ESCOLARIDADE	Tem 2 colunas e possui o código e o nome de níveis de escolaridade possíveis.
TAB_ESPECIALIZACAO	Tem 3 colunas e possui o código e o nome de cursos de especialização.
TAB_ESTADO	Tem 4 colunas e tem a função de informar o código e o nome dos estados brasileiros.
TAB_ETAPA_ENSINO	Tem 3 colunas e tem a função de informar o código e as etapas de ensino existentes.

TAB_ETAPAS_HR_COINCIDENTES	Tem 2 colunas e tem a função de informar etapas com horários coincidentes.
TAB_FORMA_INGRESSO_ALUNO	Tem 2 colunas e tem a função de informar formas de ingresso do aluno em escola federal.
TAB_GEMEOS	Tem 5 colunas e tem a função de informar os códigos dos alunos que são gêmeos.
TAB_IES	Tem 6 colunas e tem a função de informar o código, nome e outros dados de instituições de ensino superior.
TAB_LINGUA_INDIGENA	Tem 2 colunas e tem a função de informar o código e o nome de línguas indígenas.
TAB_MAT_REMANEJADA	Tem 4 colunas e tem a função de informar as matrículas que foram remanejadas do ano anterior.
TAB_MATRICULA	Tem 13 colunas e tem a função de informar dados de matrícula.
TAB_MAT_TRANSPORTE	Tem 5 colunas e tem a função de informar o código do tipo de veículo utilizado no transporte escolar para matrículas de alunos que o utilizam.
TAB_MOD_ENSINO	Tem 2 colunas e tem a função de informar o código e a descrição das modalidades de ensino.
TAB_MOD_ETAPA_ENS	Tem 2 colunas e tem a função de informar as combinações entre modalidades e etapas de ensino.
TAB_MODULO	Tem 4 colunas e tem a função de informar os

	módulos do sistema Educacenso.
TAB_MOVIMENTO_RENDIMENTO	Tem 9 colunas e tem a função de informar o rendimento dos alunos (aprovado, reprovado, concluinte, etc).
TAB_MUNICIPIO	Tem 10 colunas e tem a função de informar o código, o nome e outros dados de municípios brasileiros.
TAB_NECESSIDADE_INCOMPATIVEL	Tem 2 colunas e tem a função de informar necessidades incompatíveis.
TAB_NIS_ALUNO	Tem 3 colunas e tem a função de informar o código do aluno e seu respectivo Número de Identificação Social (NIS).
TAB_NIS_DOCENTE	Tem 3 colunas e tem a função de informar o código do docente e seu respectivo NIS.
TAB_NIVEL_USUARIO	Tem 6 colunas e tem a função de informar o nível de acesso ao sistema (leitor, executor, supersuário) de um usuário no sistema Educacenso.
TAB_ORGAO_EMISSOR	Tem 2 colunas e tem a função de informar o código e o nome dos tipos de órgão emissores de identidade.
TAB_PAIS	Tem 2 colunas e tem a função de informar o código e o nome dos países.
TAB_POS_GRADUACAO	Tem 2 colunas e tem a função de informar o código e o nome de cursos de pós-graduação.
TAB_RECURSO	Tem 2 colunas e tem a função de informar o código e o nome de recursos necessários para uso de alunos em provas aplicadas pelo INEP.

TAB_RECURSO_NECESSIDADE	Tem 2 colunas e tem a função de informar a combinação entre recursos e necessidades.
TAB_SISTEMA	Tem 2 colunas e tem a função de informar o código e o nome de sistemas.
TAB_SUBAREA_TIPO_ATIVIDADE	Tem 3 colunas e tem a função de informar a subárea da área dos tipos de atividades possíveis.
TAB_TIPO_AEE	Tem 2 colunas e tem a função de informar o código e as atividades do Atendimento Educacional Especializado (AEE).
TAB_TIPO_ATIVIDADE	Tem 4 colunas e tem a função de informar o código e o tipo de atividades para turmas de Atividade Complementar.
TAB_TIPO_CONTRATACAO	Tem 2 colunas e tem a função de informar o código e os tipos de contratações possíveis para docentes.
TAB_TIPO_DEPENDENCIA	Tem 3 colunas e tem a função de informar o código e o tipo de dependências que são possíveis em uma escola.
TAB_TIPO_ENTIDADE_PERFIL	Tem 8 colunas e tem a função de informar os tipos de entidades que se associam aos perfis.
TAB_TIPO_EQUIPAMENTO	Tem 2 colunas e tem a função de informar o código e o nome de equipamentos que podem existir em uma escola.
TAB_TIPO_LOCAL_FUNC	Tem 2 colunas e tem a função de informar o código e o nome dos locais de funcionamento possíveis para uma escola.
TAB_TIPO_LOCALIZACAO	Tem 3 colunas e tem a função de informar o

	código e os tipos de localização possíveis para escolas.
TAB_TIPO_MANTENEDORA	Tem 2 colunas e tem a função de informar os tipos de entidades mantenedoras de escolas privadas.
TAB_TIPO_MOVIMENTO	Tem 3 colunas e tem a função de informar os tipos de movimentos possíveis para um aluno.
TAB_TIPO_MOV_REND	Tem 2 colunas e tem a função de informar o código e a descrição dos tipos de movimento e rendimento.
TAB_TIPO_NECESSIDADE	Tem 3 colunas e tem a função de informar o código e a descrição dos tipos de necessidades (deficiências).
TAB_TIPO_NIVEL_ACESSO	Tem 2 colunas e tem a função de informar os tipos de nível de acesso que um usuário tem no sistema.
TAB_TIPO_TRANSPORTE	Tem 2 colunas e tem a função de informar o código e a descrição dos tipos de transporte.
TAB_TIPO_TURMA	Tem 2 colunas e tem a função de informar o código e a descrição dos tipos de turma.
TAB_TURMA	Possui 18 colunas com dados a respeito do código, nome, modalidade e etapa de ensino, horário e status da turma e outros.
TAB_TURMA_DIAS_SEMANA	Possui 4 colunas e tem a função de informar os dias da semana em que uma turma funciona.
TAB_TURMA_REMANEJADA	Possui 3 colunas e tem a função de informar quais turmas foram remanejadas do ano anterior.

TAB_TURMA_TIPO_AEE	Possui 4 colunas e tem a função de informar qual os tipos de Atividades de Educação Especializada (AEE) uma turma oferece.
TAB_TURMA_TIPO_ATIVIDADE	Possui 6 colunas e tem a função de informar os tipos de atividade complementar que uma turma oferece.
TC_CARTORIO	Tem a função de informar dados de cartórios brasileiros.